Université de Montréal

Direction des bibliothèques

AVIS

Ce document a été numérisé par la Division de la gestion des documents et des archives de l'Université de Montréal.

L'auteur a autorisé l'Université de Montréal à reproduire et diffuser, en totalité ou en partie, par quelque moyen que ce soit et sur quelque support que ce soit, et exclusivement à des fins non lucratives d'enseignement et de recherche, des copies de ce mémoire ou de cette thèse.

L'auteur et les coauteurs le cas échéant conservent la propriété du droit d'auteur et des droits moraux qui protègent ce document. Ni la thèse ou le mémoire, ni des extraits substantiels de ce document, ne doivent être imprimés ou autrement reproduits sans l'autorisation de l'auteur.

Afin de se conformer à la Loi canadienne sur la protection des renseignements personnels, quelques formulaires secondaires, coordonnées ou signatures intégrées au texte ont pu être enlevés de ce document. Bien que cela ait pu affecter la pagination, il n'y a aucun contenu manquant.

NOTICE

This document was digitized by the Records Management & Archives Division of Université de Montréal.

The author of this thesis or dissertation has granted a nonexclusive license allowing Université de Montréal to reproduce and publish the document, in part or in whole, and in any format, solely for noncommercial educational and research purposes.

The author and co-authors if applicable retain copyright ownership and moral rights in this document. Neither the whole thesis or dissertation, nor substantial extracts from it, may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms, contact information or signatures may have been removed from the document. While this may affect the document page count, it does not represent any loss of content from the document.

Université de Montréal

Heuristiques efficaces pour l'optimisation de la performance des systèmes séries-parallèles

par

Mohamed Ouzineb

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures

en vue de l'obtention du grade de

Docteur ès sciences (Ph.D.)

en informatique

Mars 2009

© Mohamed Ouzineb, 2009

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée:

Heuristiques efficaces pour l'optimisation de la performance des systèmes séries-parallèles

présentée par:

Mohamed Ouzineb

a été évaluée par un jury composé des personnes suivantes:

Pierre L'Écuyer

(président-rapporteur)

Michel Gendreau

(directeur de recherche)

Mustapha Nourelfath

(co-directeur)

Jacques Ferland

(membre du jury)

Eric Châtelet

(examinateur externe)

Richard Mackenzie

(représentant du doyen de la F.A.S.)

Thèse acceptée le:

22 juin 2009

Résumé

Les travaux de cette thèse portent sur la conception optimale des systèmes de production dont les composantes sont assujetties à des défaillances aléatoires. Nous nous intéressons au développement d'algorithmes avancés pour différents problèmes d'optimisation de systèmes de production utilisant la redondance comme technique d'amélioration de la performance. Le système est ainsi composé d'un ensemble de sous-systèmes en série, et chaque sous-système contient un nombre de composantes en parallèle. La méthodologie adoptée développe des modèles de conception optimale, des méthodes d'évaluation de la performance et des algorithmes de résolution efficaces et robustes à base de métaheuristiques. Les contributions de la thèse se situent au niveau du développement d'algorithmes de résolution de quatre problèmes de conception optimale des systèmes de production appartenant à la famille de problèmes combinatoires difficiles.

Le premier problème concerne l'allocation optimale de la redondance des systèmes séries-parallèles multi-états homogènes. On considère qu'un système et ses composantes peuvent occuper plusieurs états avec différents niveaux de performances, allant du fonctionnement parfait à la défaillance complète. Dans chaque sous-système, des composantes identiques doivent être sélectionnées pour atteindre un niveau donné de fiabilité (ou de disponibilité dans le cas d'un système réparable). Chaque composante est caractérisée par sa fiabilité, sa performance et son coût. La fiabilité du système multi-états (SME) est mesurée par la probabilité que la performance de ce système soit supérieure ou égale à la demande (niveau de performance requis par le client). Cette demande est représentée par une courbe de charge cumulative. L'objectif est de minimiser le coût de

la structure du système série-parallèle sous une contrainte de fiabilité. Afin d'estimer la fiabilité d'un système multi-état série-parallèle, nous utilisons une méthode basée sur la fonction génératrice universelle (FGU). Ce choix est motivé par le fait que cette méthode est assez rapide pour être utilisée pour des problèmes d'allocation optimale de la redondance des SME de grandes tailles. L'approche de résolution combine une idée originale de partitionnement de l'espace global de recherche en sous-espaces disjoints, et une heuristique efficace de recherche avec tabous appliquée à chacun de ces sous-espaces. Les résultats obtenus sont comparés à ceux obtenus par l'algorithme GENITOR, un algorithme génétique considéré comme le plus efficace pour ce problème. En plus des problèmes tests issus de la littérature, différentes instances de plus grandes tailles sont générées aléatoirement. L'étude comparative montre l'efficacité de notre approche tant au niveau du temps de calcul qu'au niveau de la qualité des solutions.

Le second problème concerne l'allocation optimale de la redondance des systèmes séries-parallèles multi-états hétérogènes. Ce problème diffère du premier par le fait que des composantes non identiques peuvent être dans chaque sous-système. En levant cette contrainte d'homogénéité de structure, la méthode que nous avons proposé pour le preinier problème s'est avérée coûteuse en temps de calcul. Cette limite est due au nombre élevé de sous-espaces résultant du partitionnement de l'espace global de recherche dans le cas de structures hétérogènes. Pour cela, nous avons proposée une approche qui, en plus de partitionner l'espace global et d'appliquer la recherche avec tabous à des sous-espaces, utilise l'algorithme GENITOR pour sélectionner les meilleurs sous-espaces de recherche. Cette approche est appelée PE/TG (désignant Partitionnement de l'Espace/Tabou-Génétique). Une étude comparative avec les meilleurs résultats publiés dans la littérature (GENITOR) montre l'efficacité de PE/TG tant au niveau du temps de calcul qu'au niveau de la qualité des solutions.

L'approche PE/TG est ensuite validée sur le problème d'allocation de la redondance (PAR) des systèmes binaires. On considère que le système et ses composantes ne peuvent occuper que deux états : fonctionnement parfait et défaillance complète. L'objectif est de maximiser la fiabilité du système série-parallèle sous des contraintes de coût et de poids. Les résultats numériques obtenus pour 33 problèmes tests, existants

dans la littérature du PAR, montrent que l'algorithme PE/TG est très compétitif par rapport aux autres approches publiées dans la littérature. Enfin, l'approche PE/TG est appliquée au problème de la planification des extensions des systèmes séries-parallèles multi-états homogènes. Dans ce problème, l'horizon de l'étude est divisé en plusieurs périodes. À chaque période, la demande est représentée par une courbe de charge cumulative. Pendant la durée de vie du système, la demande peut augmenter et la capacité du système peut devenir insuffisante pour satisfaire la demande. Des composantes peuvent alors être ajoutées au système afin d'augmenter sa capacité. L'objectif du problème de la planification des extensions est de minimiser la somme des coûts d'investissements sur l'horizon de l'étude, tout en satisfaisant les contraintes de disponibilité du système à chaque période. Les résultats numériques obtenus pour 6 problèmes tests issus de la littérature montrent l'efficacité de l'algorithme PE/TG par rapport aux meilleurs résultats publiés.

Mots clés. Fiabilité et productivité, Systèmes multi-états, Optimisation, Métaheuristiques, recherche avec tabous, Algorithme génétique

Abstract

This dissertation deals with the design of optimal production systems where the components are subject to random failures. We are interested in the development of efficient algorithms for solving production system optimization problems using redundancy as a technic for improving performance. The system consists of sub-systems in series, and for each sub-system, multiple component choices are used in parallel. The methodology proposed in this thesis develops optimal design models, methods for evaluating the performance of production systems and efficient and robust resolution algorithms based on meta-heuristics. The contributions of this dissertation are the development of algorithms for solving four difficult combinatorial problems.

The first problem concerns the redundancy allocation problem of homogenous series-parallel multi-state systems. Multi-state reliability modeling considers that a system and its components may experience a range of performance levels from perfect functioning to complete failure. Identical redundant components are included in order to achieve a desirable level of reliability (or availability for reparable systems). Each component is characterized by its reliability, its performance and its cost. Multi-state systems (MSS) reliability is defined by the probability that the system performance is not lower than a specific demand level which is represented as a piecewise cumulative load curve. The objective is to design the series-parallel system so that the total cost is minimized, subject to a multi-state reliability constraint. To estimate the series-parallel MSS reliability, we use the universal moment generating function (UMGF) which has been proven to be very effective for high-dimension combinatorial optimization problems. The approach, used for solving the problem, combines an original idea of partitioning

the search space into a set of disjoint subsets, and an efficient tabu search applied to each subset. The obtained results are compared with those obtained by GENITOR algorithm. This algorithm belongs to the family of genetic algorithms and is considered as the most effective algorithm for this problem. Numerical results for the test problems from previous research are reported, and larger test problems are randomly generated. Comparisons show that the proposed tabu search outperforms genetic algorithm solutions, both in terms of the solution quality and the execution time.

The second problem concerns the redundancy allocation problem for non-homogeneous series-parallel multi-state systems. This problem differs from the first by this fact that components are not identical in each sub-system. When mixing component is permitted, the problem becomes very complex due to the enormous size of the search space, and the proposed method for the first problem has been costly in the computing time. For solving the second problem, we proposed an approach based on a combination of space partitioning, tabu search and GENITOR algorithm which used to select the best subspaces. We call it space partitioning/tabu-genetic (SP/TG), SP and TG being acronyms of Space Partitioning and Tabu-Genetic, respectively. The obtained results show that the proposed approach is efficient both in terms of solution quality and computational time, as compared to existing approaches.

The approach is then validated on the redundancy allocation problem of series-parallel binary-state systems. It is assumed in binary-state reliability modeling that a system and its components may experience only two possible states : good and failed. The design goal is to select the optimal combination of components and redundancy levels so that the total reliability is maximized, subject to budget and to weight constraints. Numerical results, for the 33 test problems existing in the literature of the RAP, show that the SP/TG approach is very competitive compared to other approaches in the literature. Finally, the SP/TG approach is applied to the expansion-scheduling problem of multi-state series-parallel systems. In this problem, the study period is divided into several stages. At each stage, the demand is represented as a piecewise cumulative load curve. During the system lifetime, the demand can increase and the total productivity may become insufficient to assume the demand. To increase

the total system productivity, components are added to the existing system. The objective is to minimize the sum of costs of the investments over the study period while satisfying availability constraints at each stage. The numerical results obtained for 6 different test problems existing in the literature show the effectiveness of the algorithm SP/TG approach with the best published results.

Keywords. Redundancy allocation, Expansion planning, Multi-state systems, Series-parallel systems, Tabu search, Genetic algorithms, Space partitioning

Table des matières

1 Introduction générale et revue de la littérature 1

1.1 Motivation et contexte 1

1.2 Exemple d'un système multi-état 4

1.2.1 Description du système 4

1.2.2 Description de la demande 5

1.3 Problématique 6

1.3.1 Allocation de la redondance des systèmes séries-parallèles binaires 6

1.3.2 Le PAR des systèmes séries-parallèles multi-états (cas homogène et hétérogène) 7

1.3.3 Planification des extensions des systèmes multi-états 8

1.4 Objectifs 10

1.5 Revue de la littérature 11

1.5.1 Allocation de la redondance dans les systèmes binaires 11

1.5.2 Évaluation et optimisation des systèmes multi-états 13

1.6 La méthode d'évaluation de la disponibilité des SME 15

1.6.1 Composantes en parallèle 16

1.6.2 Composantes en série 17

1.6.3 La disponibilité des SME 18

1.7 Résumé de la méthodologie 19

1.7.1 La méthode de recherche avec tabous 20
1.7.2 Algorithmes génétiques 23
1.7.3 Partition de l'espace de recherche 25
1.7.4 Particularités de recherche avec tabous appliquée à un sous-espace 26
1.7.5 Hybridation de l'algorithme génétique et recherche avec tabous . 27
1.7.6 Calibrage des paramètres 28
1.8 Plan de la thèse 29
1.9 Conclusion 30

2 Tabu search for the redundancy allocation problem of homogenous series-parallel multi-state systems 31

2.1 Introduction 37
2.2 The redundancy allocation problem for series-parallel multi-state systems 39
2.2.1 General description of the problem and assumptions 39
2.2.2 Mathematical formulation of the problem 40
2.2.3 Availability of repairable multi-state systems 41
2.3 MSS availability estimation method 42
2.3.1 Definitions and properties of the U-function 42
2.3.2 Series-parallel MSS availability evaluation using U-functions . . . 44
2.4 The tabu search approach 46
2.4.1 Partitioning the search space 47
2.4.2 Applying tabu search to subspaces 49
2.4.3 The general algorithm 51
2.5 Test problems and numerical results 51
2.5.1 Test problems 52
2.5.2 Errata for [51, 56] 60
2.5.3 Results obtained by TS 61
2.5.4 Comparing the proposed TS with GA 62
2.5.5 More comparisons of TS and GA 65

2.6 Conclusion 66

3 A heuristic method for non-homogeneous redundancy optimization of series-parallel multi-state systems 71

3.1 Introduction 75

3.2 Problem formulation 77

3.2.1 General description 77

3.2.2 Assumptions 78

3.2.3 Notation 78

3.2.4 Mathematical model 79

3.2.5 Availability of MSS 80

3.3 Solution methodology 82

3.3.1 Partition of the search space 82

3.3.2 Tabu search 82

3.3.3 Hybrid optimization method 83

3.4 Computational results 85

3.4.1 Test problems 85

3.4.2 Parameter settings 86

3.4.3 GA re-implementation for comparison 87

3.4.4 Comparisons of the proposed SP/TG with GA 88

3.4.5 Convergence of SP/TG and GA 94

3.5 Conclusion 95

4 An efficient heuristic for reliability design optimization problems 100

4.1 Introduction 104

4.2 Two typical reliability design optimization problems 105

4.2.1 The redundancy allocation in series-parallel binary-state systems 105

4.2.2 The expansion-scheduling of series-parallel multi-state systems . 108

4.3 The space partitioning and tabu-genetic approach 116

4.3.1 Dividing the search space into a set of disjoint subspaces 116
4.3.2 Applying tabu search to each selected subspace 116
4.3.3 Hybridization of genetic algorithm and tabu search 118
4.4 Application of the proposed SP/TG approach 120
4.4.1 Application to the RAP 120
4.4.2 Application to the ESP 122
4.5 Computational results 125
4.5.1 Redundancy allocation problem (RAP) 125
4.5.2 Expansion-scheduling problem (ESP) 134
4.5.3 Convergence 139
4.6 Conclusion 141

5 Conclusion générale **143**

Liste des tableaux

1.1 Un exemple de croisement avec coupures aux 3ème et 6ème éléments. . . 25

2.1 Data of the system elements available in the market [56] 54

2.2 Parameters of the cumulative load demand curve [56] 54

2.3 Parameters of the cost function with discounts [56] 54

2.4 Data of the system elements available in the market [50] 55

2.5 Parameters of the cumulative load demand curve [50] 56

2.6 Data of the system elements available in the market [51] 57

2.7 Parameters of the cumulative load demand curve [51] 57

2.8 Randomly generated data for the new problem 59

2.9 Size of the search space, number of subspaces and parameters values for each instance 60

2.10 Results obtained by genetic algorithm as published in [50, 51, 56] and corrected values of $A(X, J)$ 61

2.11 Results obtained by TS 62

2.12 Solutions obtained by GA 63

2.13 The comparison results 65

2.14 The results obtained by GA and TS for different running times 65

3.1 An example illustrating the 2-point crossover procedure 84

3.2 Search space size and disjoint subspaces (maximum of 10 components in parallel) 86

3.3 SP/TG parameters values 87

3.4 GA parameters values 87

3.5 SP/TG best solutions over ten runs 89

3.6 GA best solutions over ten runs 89

3.7 The comparison results (the best costs and running times among ten runs are in bold 90

3.8 The comparison results in terms of the mean values (ten trials) 91

3.9 The comparison results in terms of the worst values (among ten trials) . 92

3.10 Comparison of mean SP/TG with best GA performances over 10 seeds . 93

3.11 Standard deviations of SP/TG and GA 94

4.1 Data for RAP test problems [20] 126

4.2 Results of the best solutions obtained by SP/TG 129

4.3 Comparison of the best solutions among heuristics 130

4.4 Maximum reliability, average reliability, standard deviation and average evaluation of the SP/TG algorithm 134

4.5 The best solution found by SP/TG algorithm among the 10 runs ($Str_0 = Str_0^*$) 137

4.6 The best solution found by SP/TG algorithm among the 10 runs ($Str_0 = \emptyset$) 138

4.7 Total costs for the proposed and existing algorithms for the ESP 138

4.8 Minimum cost, average cost, standard deviation and average evaluation of the SP/TG algorithm 139

Liste des figures

1.1 Un système de transport de charbon d'une charbonnière vers une centrale électrique [50] 4

1.2 Un exemple de courbe de charge cumulative par morceaux 5

1.3 Structure initiale 9

1.4 Structure à la période 1 9

1.5 Structure à la période 2 10

1.6 Hybridation de l'algorithme génétique et la recherche avec tabous 29

2.1 Comparison of the best TS cost with the best GA cost 64

2.2 Convergence curves for GA and TS for lev4-(4/6)-3 67

2.3 Convergence curves for GA and TS for lis4-(7/11)-4 68

2.4 Convergence curves for GA and TS for lev5-(4/9)-4 69

2.5 Convergence curves for GA and TS for ouz6-(4/11)-4 70

3.1 Convergence curves for GA and SP/TG for lev4-(4/6)-3 96

3.2 Convergence curves for GA and SP/TG for lev5-(4/9)-4 97

3.3 Convergence curves for GA and SP/TG for lis4-(7/11)-4 98

3.4 Convergence curves for GA and SP/TG for ouz6-(4/11)-4 99

4.1 Initial structure of MSS 111

4.2 MSS structure for the first period 112

4.3 MSS structure for the second period 112

4.4 General algorithm 119

4.5 Example of 2-point crossover procedure 125

4.6 An example of convergence curve for the RAP 140

4.7 An example of convergence curve for the ESP 141

Remerciements

C'est avec la plus profonde gratitude que je souhaite remercier M. Michel Gendreau, directeur de recherche et M. Mustapha Nourelfath, co-directeur de recherche, pour leur aide précieuse tant didactique que financière. Leurs idées originales et leurs nombreux conseils m'ont guidé tout au long de ce travail et m'ont permis d'approfondir mes connaissances dans les domaines de la recherche opérationnelle et du génie industriel.

Je remercie également l'ensemble des membres du Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, pour leur accueil, leur soutien et leur gentillesse. Aussi, je remercie tous mes professeurs à l'Université de Montréal et tous ceux que j'ai côtoyés durant ces cinq années au sein du département d'informatique et recherche opérationnelle à l'Université de Montréal.

Je souhaite aussi exprimer toute ma reconnaissance envers les membres de jury de thèse qui m'ont accordé de leur temps et m'ont fait l'honneur de juger les travaux présentés dans cette thèse. Je pense notamment à :
Monsieur Pierre L'Écuyer, Professeur au DIRO président du jury,
Monsieur Jacques Ferland, Professeur au DIRO membre du jury,
Monsieur Éric Châtelet, Professeur à l'université de technologie de Troyes, France, examinateur externe.

Finalement, je veux remercier du fond du coeur mes parents et ma famille pour leur soutien financier et encouragements au long de mes études, ma femme Hanane pour sa patience et surtout sa compréhension pendant mes absences (soirées, week-ends, jour fériés, ...) et ma fille Youssra dont le sourire vaut mille mots.

Chapitre 1

Introduction générale et revue de la littérature

1.1 Motivation et contexte

Les systèmes de production peuvent supporter plusieurs types de défaillances durant leur cycle de vie. La défaillance d'une composante peut causer une panne totale du système, ou bien peut mener simplement à une diminution dans la production. Dans un contexte industriel, la défaillance d'un équipement est un événement indésirable souvent lourd de conséquences aussi bien au niveau humain qu'au niveau économique. Ces conséquences engendrent d'énormes pertes (perte de production, perte de temps, perte financière, ...) et obligent d'importants investissements pour la remise en état des structures défaillantes. Pour être à la mesure d'une telle situation, plusieurs stratégies ont été proposées pour améliorer la sûreté de fonctionnement des systèmes de production.

La sûreté de fonctionnement est l'aptitude d'une entité à satisfaire une ou plusieurs fonctions requises dans des conditions données [83]. Parmi les caractéristiques fondamentales de la sûreté de fonctionnement on trouve la fiabilité, la disponibilité et la maintenabilité. Si ces systèmes sont réparables alors la disponibilité est la mesure la plus appropriée pour décrire leur endurance. L'objectif de la sûreté de fonctionnement

est d'assurer la continuité du service d'un système industriel.

Dans cette optique, la tolérance aux fautes occupe une place importante dans l'amélioration de la performance des systèmes [2]. La tolérance aux fautes est la capacité d'un système à continuer de délivrer un service dans les limites de ses spécifications, malgré l'occurrence de défaillances au niveau de l'une de ses composantes. Initialement élaborées pour palier essentiellement aux éventuels dysfonctionnements matériels des systèmes. Les techniques de tolérance aux fautes ont été développées dans les domaines de l'aérospatiale et de l'armée, par la suite dans l'environnement industriel et commercial. Souvent on ne peut imposer au dispositif de fonctionner toujours sans défaillances, mais on veut seulement que les dysfonctionnements probables ne causent que des dommages modérés. Cette tolérance aux fautes est un des aspects de la fiabilité. En effet, dès le début des années 50, le développement de ces techniques a permis de concevoir des systèmes plus fiables [3]. Le concept de tolérance aux fautes exploite, entre autres, les avantages que procure la redondance [4]. Le développement de la redondance se fonde sur la théorie de la duplication d'architectures pour assurer le fonctionnement du système, en cas de panne de l'une des composantes [64, 65]. Dupliquer une ressource stratégique d'une chaîne de production ou mettre en place des plans de maintenance curative ou préventive sont deux approches différentes qui permettent la garantie d'un niveau de performance d'un système (chaîne de production ou produits fabriqués). Dans le cas de produits multi-composantes, pour satisfaire les besoins des clients en terme de fiabilité, il faut pouvoir garantir les niveaux de redondance et déterminer les fiabilités à allouer aux composantes.

En résumé, la préoccupation principale de chaque chef d'entreprise est de diminuer les coûts et augmenter la performance de son système de production. Les entreprises doivent ainsi développer des moyens de production fiables et flexibles, maintenir un niveau élevé de la qualité des produits, optimiser et rationaliser l'utilisation des équipements et réduire les coûts de conception et d'exploitation de l'ensemble du système de production. Face à une telle demande croissante pour des systèmes très fiables, plus sécuritaires et moins chers, plusieurs techniques sont généralement utilisées, aussi bien à la phase de conception que pendant l'exploitation, pour accroître la

performance (fiabilité, disponibilité, taux de production, etc.) de tels systèmes. Parmi les techniques utilisées en conception, on retrouve l'augmentation du niveau de redondance, l'introduction de stocks tampons entre les machines, et l'augmentation de la fiabilité des composantes du système. Les techniques utilisées pendant l'exploitation incluent la mise en place des plans de maintenance préventive, et l'optimisation de la planification et l'ordonnancement des activités.

C'est dans ce contexte que les travaux de ce projet vont porter sur la conception des systèmes de production dont les composantes sont assujetties à des défaillances aléatoires. Dans notre étude, une caractéristique importante retenue réside dans la possibilité de modéliser le système et ses composantes selon la théorie de la fiabilité des systèmes *multi-états* (SME).

Il existe plusieurs situations en pratique dans lesquelles un système de production doit être considéré comme un SME : (i) un système formé de différentes machines qui ont un effet cumulatif sur la production est un SME dont le taux de production dépend de la disponibilité de ces machines ; (ii) le taux de production d'un système peut diminuer avec le temps à cause des dégradations graduelles des différentes composantes (fatigue, défaillances partielles, chocs aléatoires, etc.) ou des conditions ambiantes. La fiabilité des SME considère qu'un système et ses composantes peuvent occuper plusieurs états avec différents niveaux de performances, allant du fonctionnement parfait à la défaillance complète. Par contre, dans la représentation traditionnelle de la fiabilité des systèmes dits binaires, seulement deux états sont admis : fonctionnement nominal et défaillance complète. Une modélisation du système de production comme un SME permet de tenir compte de ses différents fonctionnements dégradés. La théorie de la fiabilité des SME représente un champ de recherche relativement récent, en pleine émergence et qui est à la jonction de la fiabilité des systèmes binaires et de l'analyse de performance [9, 38, 49, 52, 85]. Bien que la modélisation d'un système de production comme un SME soit plus complexe et conduise à des calculs plus coûteux par rapport au cas binaire, elle est plus proche de la réalité industrielle.

1.2 Exemple d'un système multi-état

1.2.1 Description du système

Dans l'exemple ci-dessous (figure 1.1) proposé par Levitin dans [50], on considère un système de transport du charbon vers une centrale électrique. Ce système est constitué de cinq sous systèmes connectés en série :

1. Chargeurs primaires (primary feeder) : chargent le charbon de la charbonnière vers les convoyeurs primaires.
2. Convoyeurs primaires : transportent le charbon vers les élévateurs.
3. Élévateurs : soulèvent le charbon jusqu'au niveau du brûleur.
4. Chargeurs secondaires : chargent les convoyeurs secondaires.
5. Convoyeurs secondaires : transportent le charbon vers les chaudières.

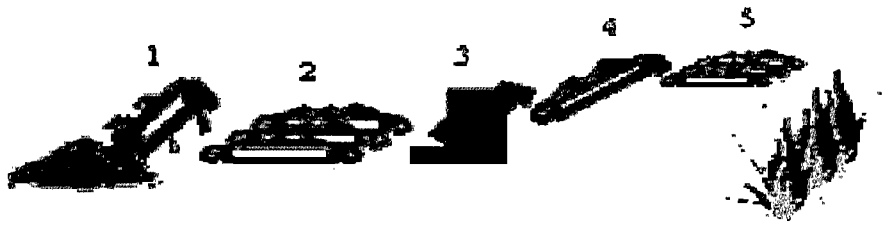


FIG. 1.1 – Un système de transport de charbon d'une charbonnière vers une centrale électrique [50]

La performance d'une composante est le rapport entre la capacité totale requise et la capacité de cette composante. Par exemple, si la capacité requise est de 125 tonnes/heure et la capacité d'un élévateur est de 100 tonnes/heure, alors la performance de ce dernier est de 80%.

Ce système de production est considéré comme un système série-parallèle multi-états. En effet, il est formé de cinq sous-systèmes en série, et chaque sous-système est un ensemble de composantes parallèle qui ont un effet cumulatif sur la production. Son taux de production dépend de la disponibilité de ces composantes. Il peut occuper plusieurs états avec différents niveaux de performances, allant du fonctionnement parfait à la

défaillance complète. Un fonctionnement dégradé correspond par exemple aux pannes d'un des convoyeurs primaires et un des chargeurs secondaires : dans ce cas, même si les taux de production des sous-systèmes 2 et 4 seront diminués, le système continuera à fournir le service demandé avec une performance amoindrie.

1.2.2 Description de la demande

La demande est prédite par une courbe de charge cumulative par morceaux (figure 1.2). Dans l'exemple d'application précédent, on suppose que la période totale de production est divisée en trois périodes : dans 20% de la période totale, la demande est de 100% (125 tonnes/heure) ; dans 30% de la période totale, la demande est de 80% (100 tonnes/heure) ; pendant les 50% restantes la demande est de 40% (50 tonnes/heure). La fiabilité (ou la disponibilité dans le cas de systèmes réparables) est mesurée par la probabilité que sa performance soit supérieure ou égale à la demande.

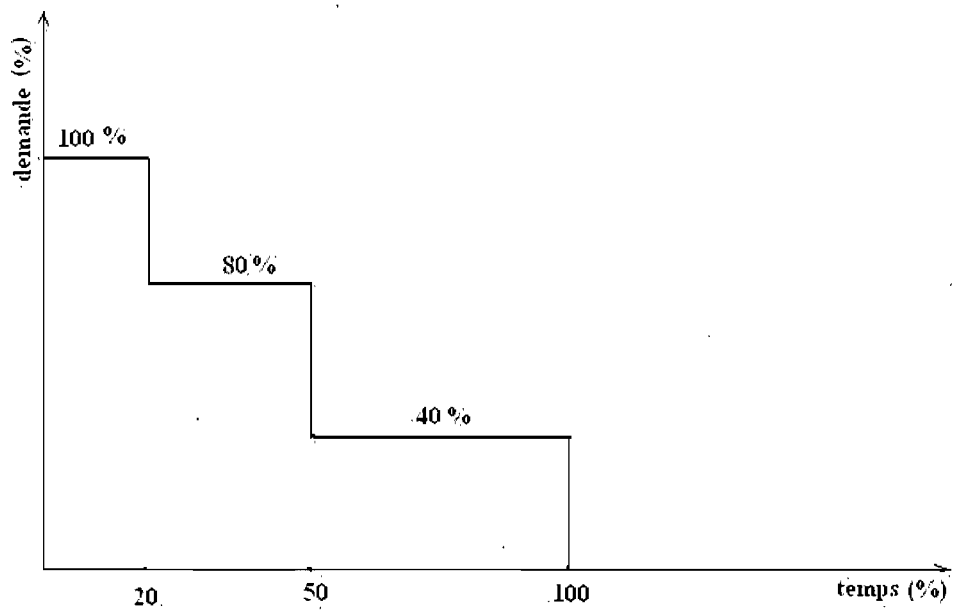


FIG. 1.2 – Un exemple de courbe de charge cumulative par morceaux

1.3 Problématique

Les travaux de cette thèse s'inscrivent dans une problématique générale d'optimisation de la conception des systèmes séries-parallèles, en tenant compte du fait que les composantes sont assujetties à des défaillances aléatoires. Les caractéristiques opératoires de ces composantes se dégradent à l'usage. Les systèmes étudiés utilisent la redondance comme technique d'amélioration de la performance. Un manque de composantes redondantes pourrait mettre la capacité du système de production à répondre à la demande en péril. De même que l'utilisation de plusieurs composantes en parallèle engendre des coûts d'installation onéreux et possiblement inutiles. Nous nous intéresserons particulièrement au développement d'algorithmes avancés pour les problèmes de conception optimale suivants :

1. Le problème d'allocation de la redondance (PAR) des systèmes séries-parallèles binaires.
2. Le PAR des systèmes séries-parallèles multi-états homogènes.
3. Le PAR des systèmes séries-parallèles multi-états hétérogènes.
4. Le problème de la planification des extensions des systèmes séries-parallèles multi-états.

Ces quatre problèmes sont d'une importance cruciale dans la conception des systèmes de production. Avant de présenter nos objectifs spécifiques, nous allons commencer par présenter la problématique relative à chacun de ces problèmes d'optimisation.

1.3.1 Allocation de la redondance des systèmes séries-parallèles binaires

Ce problème est d'une extrême importance pour la conception des systèmes ayant des exigences en matière de fiabilité, comme c'est le cas de nombreux systèmes industriels (systèmes électroniques, systèmes de télécommunication, systèmes de puissance, systèmes de production, etc.). Pour tous ces systèmes, la fiabilité peut être considérée comme une mesure importante dans la conception. D'une manière générale, la concep-

tion d'un système nécessite le choix de technologies de composantes parmi une liste disponible dans le marché. Ces technologies de composantes sont caractérisées par leur fiabilité, coût, performance, poids, etc. Une stratégie est alors requise pour identifier la combinaison optimale des technologies à implanter dans le système, ainsi que le niveau de redondance dans chaque sous-système. L'objectif est de maximiser la fiabilité sous des contraintes de coût et de poids.

Ce problème est "difficile" à résoudre compte tenu du grand nombre de solutions possibles. En fait, l'analyse de la complexité théorique de calcul a montré que ce problème était NP-difficile [14], ce qui implique que la résolution exacte d'instances de taille substantielle, susceptibles d'être rencontrées dans des applications réelles, demande en général des temps de calcul considérables. La problématique abordée consiste à mettre au point une méthode de calcul approchée (une heuristique). Cette méthode devra être caractérisée par une très bonne qualité des solutions obtenues et un temps de calcul relativement court.

1.3.2 Le PAR des systèmes séries-parallèles multi-états (cas homogène et hétérogène)

Le PAR des systèmes séries-parallèles *multi-états* est plus récent que la version binaire du problème. La plupart des approches existantes dans la littérature du PAR ne tiennent pas compte des différents niveaux de dégradation du système. La problématique abordée ici concerne l'allocation optimale de la redondance des systèmes séries-parallèles multi-états. On considère que pour chacune des composantes, il existe plusieurs versions ou technologies dans le marché. Une composante est caractérisée par sa fiabilité (ou sa disponibilité dans le cas de systèmes réparables), sa performance et son coût. La fiabilité (ou la disponibilité) du SME est mesurée par la probabilité que sa performance soit supérieure ou égale à la demande qui est représentée par une courbe de charge cumulative. L'objectif est de minimiser le coût de la structure (série-parallèle) du système multi-états sous une contrainte de fiabilité (ou de disponibilité). Deux cas seront considérés : (i) cas homogène, c'est-à-dire où les composantes parallèles sont

identiques ; (ii) cas hétérogène, c'est-à-dire où ces composantes peuvent être différentes.

Afin d'estimer la fiabilité d'un système multi-état série-parallèle, nous avons besoin d'une méthode assez rapide pour être utilisée dans la résolution de problèmes d'allocation optimale de la redondance de SME de grandes tailles. De plus, vu que les méthodes d'évaluation utilisées seront basées plutôt sur la théorie des SME, les méthodes d'optimisation disponibles dans la littérature utilisant la fiabilité des systèmes binaires (pour l'évaluation de la fiabilité ou la disponibilité) ne peuvent garantir l'obtention de solutions efficaces.

Le cas *hétérogène* diffère par le fait que des composantes *non identiques* peuvent être utilisées dans chaque sous-système. Ce cas est important pour deux raisons [50] : (i) en autorisant l'utilisation de différentes versions de composantes dans le même sous-système, on peut obtenir une solution qui offre un niveau de disponibilité (ou de fiabilité) souhaitable avec un coût inférieur au coût d'une solution offerte avec des composantes identiques en parallèle ; (ii) dans la pratique, le concepteur ajoute souvent des composantes supplémentaires dans le système existant. Il peut être nécessaire, par exemple, de moderniser un système d'alimentation en énergie électrique selon la demande ou selon de nouvelles exigences de fiabilité. Certains sous-systèmes peuvent contenir des composantes dont des versions ne sont plus disponibles au moment de la modernisation. Dans ce cas, certaines composantes avec les mêmes fonctionnalités, mais avec des paramètres différents doivent composer certains sous-systèmes.

Le nombre de solutions possibles dans le cas de systèmes séries-parallèles hétérogènes augmente de façon considérable par rapport au cas homogène. Par conséquent, une nouvelle heuristique efficace (c'est-à-dire permettant d'obtenir des solutions de très bonne qualité en un temps de calcul relativement court) devra être développée.

1.3.3 Planification des extensions des systèmes multi-états

L'horizon de l'étude est divisé en plusieurs périodes. À chaque période, la demande est prédite par une courbe de charge cumulative. Pour satisfaire la demande qui aug-

mente avec le temps, il faudra accroître la capacité de production en ajoutant des composantes supplémentaires dans le système à chaque période. Les composantes ajoutées dans le système servent pour toutes les périodes à partir de la période où elles ont été ajoutées. Chaque composante peut être ajoutée à n'importe quel sous-système et à n'importe quelle période. Ces composantes sont caractérisées par leurs coûts, leurs performances et leurs disponibilités ; elles sont choisies parmi une liste de produits disponibles dans le marché. L'objectif du problème de la planification des extensions est de minimiser la somme des coûts d'investissements sur l'ensemble de l'horizon de l'étude, tout en satisfaisant les contraintes de disponibilité du système à chaque période. Nous considérons des systèmes séries-parallèles tels que les composantes ajoutées dans un sous-système donné sont de même version. La structure initiale peut être vide ou non vide. Les figures de 1.3 à 1.5 représentent un exemple de planification d'extension avec une structure initiale non vide.

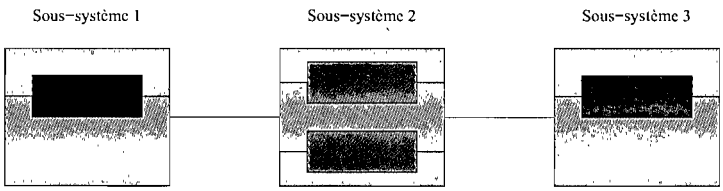


FIG. 1.3 – Structure initiale

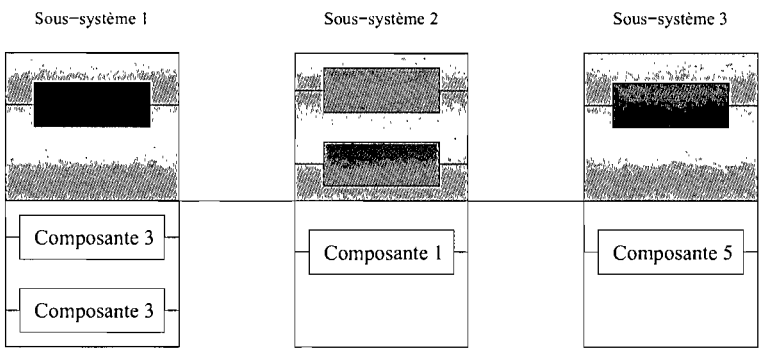


FIG. 1.4 – Structure à la période 1

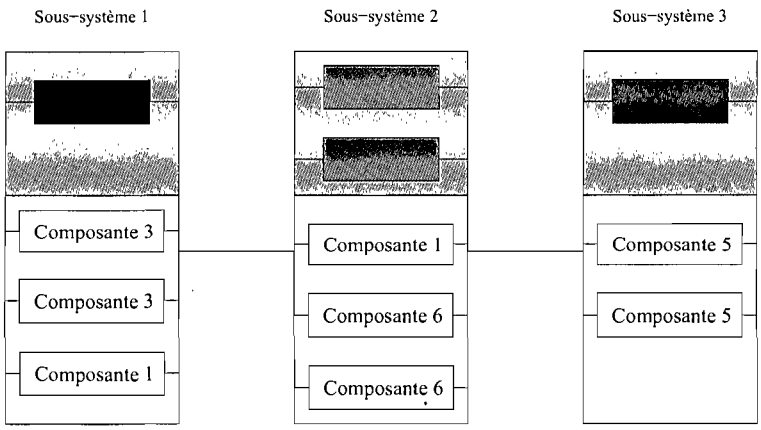


FIG. 1.5 – Structure à la période 2

1.4 Objectifs

Deux outils sont nécessaires pour résoudre chacun des quatre problèmes de conception optimale ci-dessus :

1. Une méthode d'évaluation de la performance du système de production modélisé soit comme un système binaire ou bien comme un SME : alors que la fiabilité d'un système série-parallèle binaire peut être exprimée sous une forme explicite en fonction des fiabilités de ces composantes [6], le développement d'une méthode analytique pour estimer avec une bonne précision la performance d'un SME série-parallèle est plus complexe.
2. Une méthode d'optimisation qui utilise l'outil d'évaluation : les quatre problèmes ci-dessus sont "difficiles" à résoudre compte tenu du grand nombre de solutions possible. Sachant que les méthodes exactes sont inefficaces face aux applications de tailles importantes, il va falloir développer de nouvelles méthodes approchées capables de fournir des solutions quasi-optimales, et ce en un temps assez court.

Deux objectifs spécifiques ont été ainsi poursuivis dans cette thèse :

1. Proposer des méthodes d'évaluation de la performance d'un système de production. Ces méthodes doivent être robustes et rapides et concernent également des systèmes séries-parallèles dont les composantes peuvent occuper plusieurs états

de fonctionnement distincts. La robustesse implique une estimation de la performance (fiabilité ou disponibilité) avec une précision suffisante. La rapidité signifie simplement que cette performance est calculée en un temps assez court pour permettre l'utilisation des méthodes dans les algorithmes d'optimisation.

2. Développer des heuristiques pour résoudre des problèmes de grandes tailles relatifs à la conception optimale de systèmes de production binaires et multi-états. Ces problèmes portent sur l'allocation optimale de la redondance et la planification des extensions. De façon spécifique, on considère des problèmes de conception optimale relatifs à l'allocation de la redondance des systèmes binaires et des systèmes multi-états dans les cas homogène et non homogène, et à la planification des extensions pour un système série-parallèle multi-états.

1.5 Revue de la littérature

Loin d'être exhaustive, cette revue de littérature sera classée en deux catégories : les principaux travaux sur l'allocation de la redondance dans les systèmes *binaires*, et ceux portant sur l'évaluation et l'optimisation des systèmes *multi-états*.

1.5.1 Allocation de la redondance dans les systèmes binaires

Les nombreux travaux dans ce domaine soulignent la nécessité d'intervenir en phase de conception sur la fiabilité des composantes pour garantir une fiabilité minimale au système étudié. Les premières études ont porté sur l'allocation de la fiabilité. Cette dernière est apparue dans les années 1940 et 1950 dans les domaines des communications et des transports [78]. Après les années 1960, les chercheurs se sont également intéressés à l'allocation de redondance dans les systèmes. Les variables de décision sont le nombre et le type de composantes à utiliser en parallèle. Les méthodes d'allocation de la redondance se différencient par de nombreux points tels que le type de système considéré, la fonction objectif à optimiser, les contraintes à prendre en compte, etc. En 1977, Tillman et al. [77] et en 1980, Tzafestas [79] ont publié des états de l'art sur les

techniques d'optimisation de la fiabilité des systèmes. En 1986, Dhillon [17] a proposé quant à lui une liste de références sur l'allocation de la fiabilité et Misra [58] a publié un article sur les différents travaux d'optimisation de la fiabilité. Récemment, en 2001, Kuo et al. [43] ont réactualisé l'ouvrage de Tillman et al. [78] de 1980.

Le problème d'allocation de la redondance dans les systèmes séries-parallèles est un problème combinatoire difficile [14]. Dans le cas des systèmes binaires, le PAR a été proposé avec une seule contrainte (de coût) par Bellman et Dreyfus [8]. Ils ont supposé que chaque sous-système doit contenir seulement des composantes de même version. Fyffe et al. [20] ont ajouté la contrainte de poids au même problème et ils ont utilisé la programmation dynamique avec des multiplicateurs de Lagrange pour résoudre une seule instance, dont le système entier est constitué de 14 sous-systèmes connectés en série. À ce jour, il existe plusieurs algorithmes pour résoudre ce type du problème. Une revue de littérature générale sur l'optimisation de la fiabilité des systèmes peut être trouvée dans les références [42, 87].

Nakagawa et Miyazaki [63] ont proposé un algorithme basé sur la programmation dynamique pour résoudre 33 variantes du problème de Fyffe et al. [20]. Ils ont obtenu une solution optimale pour 30 cas, et l'algorithme n'a pas convergé vers une solution réalisable pour les 3 autres cas. Parmi les autres approches utilisées pour résoudre le même problème on trouve deux approches proposées dans [78, 88] qui sont aussi basées sur la programmation dynamique. Compte tenu de l'effort de calcul qui augmente d'une façon exponentielle avec la taille du problème et le nombre de contraintes, les approches traditionnelles restreignent généralement l'espace de solutions en affectant à chaque sous-système un seul type de technologie (c'est-à-dire que seuls des systèmes homogènes sont traités). Parmi les approches utilisant cette restriction, on retrouve la programmation dynamique dans [20, 63], et la programmation en nombre entier [12] ou mixte [39].

Pour le cas où les composantes parallèles peuvent être différentes (systèmes hétérogènes), le problème a été formulé dans [16]. Un algorithme génétique (AG) a été utilisé dans [16] pour résoudre l'instance proposé par Fyffe et al. [20] pour les 33 instances in-

troduites dans [63]. Parmi les autres algorithmes, utilisés pour résoudre les mêmes cas, on trouve un algorithme de recherche avec tabous dans [41], un algorithme de fourmis dans [54], un algorithme de recherche à voisinages variables dans [53] et une heuristique hybridant des colonies de fourmis avec l'algorithme du grand déluge étendu dans [62].

1.5.2 Évaluation et optimisation des systèmes multi-états

La plupart des livres sur la théorie de la fiabilité ont développé la fiabilité binaire qui permet d'étudier les systèmes à deux états (fonctionnement parfait ou bien panne totale). Une modélisation de type multi-états est plus complexe mais elle est plus riche en information. La revue de la littérature proposée dans cette section porte sur la modélisation, l'évaluation et l'optimisation des performances (fiabilité, disponibilité et productivité) des systèmes multi-états dont les caractéristiques opératoires sont soumises à des dégradations aléatoires. Les phénomènes de dégradation qui affectent les systèmes sont souvent dus à des conditions environnementales ou des phénomènes physiques (vibrations, usures, fatigue, chocs, etc.) [44]. Ces dégradations ont plus ou moins un impact sur leurs performances. L'étude de ces systèmes nécessite une prise en compte des différents stades de dégradation qui peuvent survenir pendant leur cycle de vie. Selon les types de structures des systèmes considérés, ces stades de dégradation peuvent être multiples [72]. Pour la modélisation de tels systèmes et de leurs composantes, on associe généralement des états à ces stades de dégradation. Il existe de nombreux états de l'art sur ce domaine [11, 76, 89] qui proposent de nombreuses classifications des problèmes d'optimisation de la disponibilité des systèmes avec des politiques de maintenance déterminées et/ou des dépendances entre composantes.

Les systèmes multi-états qu'on rencontre beaucoup dans la littérature sont les systèmes de production d'électricité où la performance des composantes est caractérisée par la capacité de production électrique, et les systèmes informatiques où la performance des composantes est caractérisée par la vitesse de traitement des donnés. Pour un SME, l'effet de panne est différent selon chaque composante avec des taux de performances différents. Cela rend l'analyse de la fiabilité pour un SME très compliquée par rapport

à un système binaire. Depuis 1970, de nombreuses recherches sur la fiabilité des SME ont été entreprises.

L'analyse de la fiabilité des systèmes multi-états s'avère plus complexe que celle des systèmes binaires [7, 19, 74]. Plusieurs méthodes d'évaluation des performances des SME ont été développées ces dernières années, la plupart de ces méthodes sont basées sur la théorie des systèmes binaires pour les prolonger pour les SME, par exemple les approches markoviennes [13, 86] sont utilisées dans [10] pour calculer la fiabilité des systèmes multi-états. Puisque la méthode de Markov peut seulement s'appliquer à des SME de petites tailles (car le nombre d'états d'un système croît strictement avec la hausse du nombre de composantes du système) [5, 18, 72], et l'approche de la fonction de structure prend beaucoup de temps et ne permet pas d'analyser le comportement dynamique des SME [7, 19], il est nécessaire de développer de nouvelles approches capables de calculer la fiabilité de SME de grandes tailles, et qui soient plus efficaces que les techniques traditionnelles.

En 1986, Ushakov a développé une approche d'évaluation de la fiabilité des SME [80], basée sur la technique de «fonction génératrice universelle» (FGU). La FGU constitue une extention de la fonction de génération des moments classique [75]. L'approche de Levitin [49] basée sur la FGU s'avère rapide, ce qui permet son utilisation dans des problèmes difficiles d'optimisation des SME. Dans la démarche d'évaluation des indices de fiabilité, les problèmes traités par Levitin considèrent que les probabilités d'occurrence des états des SME sont préalablement connues et que les ressources sont disponibles pour la maintenance. Dans la littérature, les configurations structurelles des SME sont variées. Plusieurs travaux étudient les configurations multi-états unitaires, séries, parallèles ou k parmi n [38, 51, 72]. Dans cette thèse nous nous intéressons aux SME série-parallèles.

Le problème d'allocation de la redondance dans les SME série-parallèles (cas statique) a été introduit pour la première fois dans [81], où la technique FGU a été utilisée pour calculer la disponibilité. Suivant ces travaux les chercheurs ont utilisé des métaheuristiques pour trouver une configuration optimale pour un SME série-parallèle,

par exemple les auteurs des articles [46, 49, 55] ont utilisé des algorithmes génétiques pour résoudre ce type de problèmes. Les seuls autres algorithmes existant sont l'algorithme de fourmis [68] et une heuristique dans [73].

Le problème de planification optimale des extensions multi-périodes est un problème récent, formulé dans [45], où l'AG a été utilisé pour résoudre une seule instance qui a été proposée dans [50]. Le seul autre algorithme existant est un algorithme basé sur les colonies de fourmis [67].

Nous avons constaté qu'il y a très peu de travaux sur l'optimisation des coûts et de la performance des systèmes multi-états à cause de la difficulté liée au calcul de la disponibilité de ces systèmes. À notre connaissance, la plupart des approches de résolution qui ont été développées ces dernières années pour les SME sont basées sur la FGU pour calculer la disponibilité des systèmes étudiés. Cette fonction a montré sa rapidité même pour les systèmes de plus grande taille, mais elle se limite à :

1. examiner des systèmes séries-parallèles ou parallèles-séries ;
2. supposer l'indépendance entre les composantes ;
3. considérer que la demande est prédite par une courbe de charge cumulative par morceau.

1.6 La méthode d'évaluation de la disponibilité des SME

Le nombre des états du SME est considéré fini, alors on peut identifier à chaque instant t les différents états du système $\{1, 2, 3, ..., M\}$, où 1 est le plus mauvais des états et M est le meilleur état. Notons par $G(t)$ la performance totale du système à l'instant t. Chaque état m est caractérisé par un niveau de performance $G_m(t)$ et une probabilité $p_m(t)$ définie par : $p_m(t) = Pr[G(t) = G_m(t)]$. Supposons que nous connaissons les états de chaque composante du système. La technique fonction de génération universelle (FGU), dite aussi U-fonction, est utilisée pour définir les états du SME série-parallèle. Elle est basée sur la définition 1 (en bas) et les quatre propriétés d'Ushakov suivantes.

Définition 1

La U-fonction d'une variable aléatoire discrète $G(t)$ est définie comme un polynôme qui représente tous les états possibles :

$$U(z) = \sum_{m=1}^{M} p_m(t) z^{G_m(t)}. \tag{1.1}$$

Le calcul de la U-fonction correspondant au système entier est donné par l'opérateur ψ qui vérifie les propriétés suivantes (dites d'Ushakov [80]) :

1. $\psi(pz^a) = pz^a$,
2. $\psi(p_1 z^{a_1}, p_2 z^{a_2}) = p_1 p_2 z^{f(a_1,a_2)}$,
3. $\psi(U_1(z), ..., U_k(z), U_{k+1}(z), ..., U_I(z)) =$
   $\psi(\psi(U_1(z), ..., U_k(z)), \psi(U_{k+1}(z), ..., U_I(z)))$ $\forall I$ *et* $\forall k$, $1 \leq k \leq I$
4. $\psi(U_1(z), ..., U_k(z), U_{k+1}(z), ..., U_I(z)) =$
   $\psi(U_1(z), ..., U_{k+1}(z), U_k(z), ..., U_I(z))$ $\forall I$ *et* $\forall k$, $1 \leq k \leq I$

où $f(a_1, a_2)$ est définie selon la nature physique des composantes.

Voici quelques exemples pour la fonction f :

- $f(a_1, a_2) = \min(a_1, a_2)$ si les composantes sont connectées en série et la performance de chaque composante est caractérisée par son taux de production.
- $f(a_1, a_2) = (\frac{1}{a_1} + \frac{1}{a_2})^{-1}$ si les composantes sont connectées en série et la performance de chaque composante est caractérisée par son temps de fonctionnement.
- $f(a_1, a_2) = a_1 + a_2$ si les composantes sont connectées en parallèle.

Pour calculer le polynôme $U(z)$ du SME série-parallèle, nous commençons par calculer le polynôme qui correspond aux composantes en parallèle puis celui qui correspond aux composantes en série.

1.6.1 Composantes en parallèle

Quand la mesure de performance est reliée à la productivité ou à la capacité du système, la performance totale du sous-système est la somme des performances de tous

ces composantes en parallèle et la fonction f utilisée dans la propriété 2 de U-fonction est : $f(a_1, a_2) = a_1 + a_2$.

La U-fonction du sous-système contenant s composantes en parallèle peut être calculée en utilisant l'opérateur suivant :

$$U_p(z) \quad = \quad \psi(U_1(z), U_2(z), ..., U_s(z)). \tag{1.2}$$

Par exemple, pour deux composantes en parallèle on a :

$$\psi(U_1(z), U_2(z)) \quad = \quad \psi(\sum_{i=1}^{n} P_i z^{a_i}, \sum_{j=1}^{m} Q_j z^{b_j}) = \sum_{i=1}^{n} \sum_{j=1}^{m} P_i Q_j z^{a_i + b_j}. \tag{1.3}$$

Les paramètres a_i et b_i sont des niveaux de performance des composantes et P_i et Q_i sont respectivement les probabilités stationnaires correspondantes alors que n et m sont les nombres de niveaux possibles.

1.6.2 Composantes en série

Quand la mesure de performance est reliée à la productivité ou la capacité du système, la fonction f utilisée dans la propriété 2 de la U-fonction est : $f(a_1, a_2) = \min\{a_1, a_2\}$.

La U-fonction du système contenant s composantes en série peut être calculée en utilisant l'opérateur suivant :

$$U_s(z) \quad = \quad \psi(U_1(z), ..., U_s(z)). \tag{1.4}$$

Pour deux composantes en série on a :

$$\psi(U_1(z), U_2(z)) \quad = \quad \psi(\sum_{i=1}^{n} P_i z^{a_i}, \sum_{j=1}^{m} Q_j z^{b_j}) = \sum_{i=1}^{n} \sum_{j=1}^{m} P_i Q_j z^{\min\{a_1, a_2\}}. \tag{1.5}$$

L'équation (1.5) signifie simplement que c'est la composante correspondant au goulot d'étranglement qui va définir le taux de production du système des deux composantes en série).

1.6.3 La disponibilité des SME

Après avoir calculé la U-fonction correspond au système multi-états, on passe à l'évaluation de sa disponibilité. Un SME est caractérisé par son évolution dans l'espace d'états où l'ensemble des états possibles E est considéré fini. Chaque état dépend de la relation entre la performance totale du système $G(t)$ et la demande $W(t)$ ($F(G(t), W(t))$ $= G(t) - W(t)$). L'ensemble E peut être divisé en deux sous-ensembles disjoints E_1 correspondant aux états qui ne répondent pas à la demande ($F(G(t), W(t)) < 0$) et E_2 correspondant aux états qui répondent à la demande ($F(G(t), W(t)) \geq 0$).

La disponibilité $A(t)$ d'un SME est la probabilité que la capacité totale du système soit supérieure ou égale à la demande à l'instant t, sa formule mathématique est donnée par :

$$A(t) = Pr(F(G(t), W(t)) \geq 0). \tag{1.6}$$

On suppose qu'il existe une distribution de défaillances stationnaire (étude dans le régime stationnaire). Cette hypothèse est acceptable à condition d'exclure certains phénomènes qui pourraient influer sur la distribution des composantes ou encore de considérer des périodes de production assez longues pour être sûr que le régime stationnaire soit une bonne approximation.

Si W exprime une demande constante, l'expression de la disponibilité $A(W)$ du SME est donnée par :

$$A(W) = \sum_{m=1}^{M} p_m \delta_{[F(G_m, W) \geq 0]}, \tag{1.7}$$

où $\delta_{[\text{vrai}]} = 1$, $\delta_{[\text{faux}]} = 0$, $p_m = \lim_{t \to \infty} p_m(t)$ et $G_m = \lim_{t \to \infty} G_m(t)$.
Les probabilités $p_m(t)$ et les capacités $G_m(t)$ sont donnés par la U-fonction correspondant au système entier (équation 1.1).
Dans le cas, d'une demande variable d'une période à l'autre, on divise la période totale T en K intervalles de temps, la durée et la demande requise pour chaque intervalle de

temps k sont respectivement T_k et W_k, $(k = 1, ..., K)$. La disponibilité du SME devient :

$$A(W) = \frac{1}{\sum_{k=1}^{K} T_k} \sum_{k=1}^{K} A(W_k) T_k. \quad (1.8)$$

1.7 Résumé de la méthodologie

Dans cette section, nous décrivons les heuristiques que nous avons développées pour résoudre les problèmes définis dans les sections précédentes. Plus particulièrement, nous utilisons deux métaheuristiques : la recherche avec tabous (RT) et l'algorithme génétique (AG). Afin de tirer profit des avantages de ces deux métaheuristiques, une réflexion particulière a été menée sur leur hybridation en se basant sur une idée originale de partitionnement de l'espace de recherche en sous-espaces disjoints.

Plusieurs définitions ont été proposées dans la littérature [25, 28, 69] pour expliquer clairement ce qu'est une métaheuristique. Une métaheuristique peut être vue comme une stratégie qui permet de guider la recherche d'une très bonne solution à un problème donné par une heuristique adaptée à celui-ci. L'algorithme de recherche avec tabous et l'algorithme génétique sont deux des métaheuristiques qui se sont montrées parmi les plus efficaces pour la résolution d'un grand nombre de problèmes d'optimisation combinatoires difficiles [15, 16, 26, 41].

Cette section est divisée en cinq parties. Les deux premières rappellent les principes et les éléments de base de la recherche avec tabous et des algorithmes génétiques. La méthode de partitionnement de l'espace global de recherche est présentée dans la troisième sous-section. La quatrième sous-section s'attarde sur les éléments qui sont spécifiques à notre RT. À la fin, on présente une hybridation de l'algorithme de RT et l'AG.

1.7.1 La méthode de recherche avec tabous

La méthode de recherche avec tabous a été introduite par Fred Glover [29] en 1986. Cette méthode a été largement utilisée pour résoudre un grand nombre de problèmes fortement combinatoires. Pour une étude complète sur la RT, nous référons le lecteur aux travaux publiés dans [22, 23, 24, 33, 34]. La RT s'appuie sur divers concepts de l'intelligence artificielle, mais elle est directement inspirée des procédures d'améliorations itératives qui améliorent une solution de départ en lui appliquant une série de modifications locales. La caractéristique principale de cette méthode réside dans le fait qu'on permet que les mouvements élémentaires effectués pour passer d'une solution à la suivante puissent détériorer la valeur courante de la fonction objectif. À cause de cela, la procédure ne s'arrête pas lorsqu'elle arrive dans un optimum local, Comme cela pourrait amener la recherche à tourner en rond (cycler), la recherche avec tabous utilise une structure de mémorisation temporaire pour conserver une histoire récente de la recherche. Le plus souvent, il s'agit d'une liste des mouvements précédents ou des solutions précédentes.

Nous décrivons maintenant les principaux éléments constitutifs de la méthode de recherche avec tabous.

1.7.1.1 Voisinage, Mouvements

A chaque itération, on passe d'une solution $s \in S$ (l'ensemble des solutions admissibles) à une solution $s' \in S$ en appliquant à s un ou plusieurs types de modification. Ces modifications s'appellent des *mouvements*. L'ensemble des solutions obtenues suite à ces mouvements définit un *voisinage* de la solution s noté par $N(s)$. La solution s' représente habituellement la meilleure solution de $N(s)$. Si aucune solution du voisinage ne permet d'améliorer la fonction objectif, alors on sélectionne celle qui détériore le moins possible sa valeur. La taille de voisinage joue un rôle très important dans la complexité de l'algorithme de recherche avec tabous car la majorité du temps d'exécution est pris par l'évaluation des voisinages. Plus la taille du voisinage est grande, meilleure

est la chance de trouver une bonne solution. Mais, de l'autre côté, le temps d'exécution augmente très vite avec le nombre de voisins considérés. Puisque ce nombre dépend directement du choix des mouvements, alors ce choix a un grand impact sur la performance de l'algorithme.

1.7.1.2 Liste Taboue

Comme nous l'avons précédemment indiqué, afin de réduire le risque de cycler, la recherche avec tabous utilise une structure de mémorisation temporaire pour conserver une histoire récente de la recherche. Le plus souvent, il s'agit de la liste des mouvements précédents ou des solutions précédentes. Ces mouvements ou ces solutions seront interdits pendant un certain nombre d'itérations, qui correspond à la taille de la liste taboue.

La taille de la liste taboue est un paramètre important en ce qui concerne l'efficacité de l'heuristique. Plus cette taille est grande, plus on peut empêcher un retour dans les optimums locaux visités précédemment. En contrepartie, si la liste est trop longue, on peut assister à une stagnation de la recherche. De plus, dans de nombreux cas, le temps requis pour vérifier si un mouvement est tabou croit linéairement avec la longueur de la liste. Il n'est donc pas évident de déterminer la longueur optimale de la liste. Pour cette raison, dans de nombreux travaux, les auteurs s'intéressent à une liste taboue dont la taille varie de façon dynamique dans le but de parcourir plus efficacement l'espace des solutions.

Le réglage de la longueur de la liste taboue dépend donc essentiellement des résultats de la recherche. En général, la taille de la liste taboue doit être fixée à une valeur minimale permettant d'éviter les cycles pendant la recherche. En cas de blocage, *i.e.*, si la solution courante ne s'améliore pas, on diminue la longueur de la liste pour intensifier la recherche en autorisant de nouvelles solutions (en évitant d'interdire trop de solutions dans le voisinage de la solution courante). En contrepartie, si des solutions sont fréquemment visitées, alors on augmente la longueur de la liste, ce qui aura pour effet de diversifier la recherche.

1.7.1.3 Critères d'arrêt

Comme dans toute méthode de résolution, on doit définir un critère d'arrêt à la recherche. On peut choisir une limite sur le temps d'exécution ou sur le nombre d'itérations, mais le critère le plus courant en RT est une limite sur le nombre d'itérations depuis la dernière amélioration de la meilleure solution connue.

1.7.1.4 Algorithme standard de recherche avec tabous

La version simple de l'algorithme de recherche avec tabous peut se résumer comme suit :

Etape 1 : Initialisation

1. générer une solution initiale.
2. initialiser la liste taboue.

Etape 2 : Recherche avec tabous

Tant que le critère d'arrêt n'est pas vérifié,

1. générer un voisinage de la solution courante ;
2. sélectionner la meilleure solution du voisinage ou celle qui dégrade moins la solution actuelle en tenant compte de la liste taboue ;
3. mettre à jour la solution actuelle, le nombre d'itérations et la liste taboue.

1.7.1.5 Guidage de la recherche

Pour enrichir l'algorithme standard de recherche avec tabous, les trois concepts suivants peuvent être vus comme une utilisation de la connaissance acquise durant la recherche :

1. **Critère d'aspiration :** Il existe des situations où il serait souhaitable de choisir des solutions qui sont interdites en raison des tabous couramment actifs. En effet, ces solutions peuvent être attrayantes, car elles permettent d'améliorer la

meilleure solution connue ou de mener vers des régions qui semblent prometteuses dans l'espace de recherche. Le rôle du critère d'aspiration est de lever l'interdiction taboue et aussi de guider la recherche vers des solutions qui s'avèrent très intéressantes. Pour une étude complète sur les critères d'aspiration, nous référons le lecteur aux travaux publiés dans [32].

2. **Intensification :** Ce concept correspond à une exploration plus intense d'une région de l'espace des solutions qui semble particulièrement intéressante (souvent autour de la meilleure solution connue). L'intensification se fait généralement en fixant les valeurs de certaines variables, en augmentant la taille du voisinage exploré ou en changeant la structure des voisinages explorés.
3. **Diversification :** Il arrive parfois que la recherche se concentre dans une partie trop restreinte de l'espace des solutions. Le but de la diversification est de forcer l'exploration de nouvelles parties de l'espace de recherche. Ceci peut être accompli en utilisant une ou plusieurs des stratégies discutées en détails dans [34, 40]. Parmi ces stratégies on trouve :
   (a) la relance de la recherche à partir de solutions qui différent significativement de celles considérées jusque là ;
   (b) la perturbation de la fonction objectif en introduisant un paramètre qui pénalise les modifications effectuées de façon fréquente ;
   (c) la modification de la taille de la liste taboue.

1.7.2 Algorithmes génétiques

Les algorithmes génétiques (AGs) ont été présentés par John Holland [37] en 1975 et décrits plus en détail par Goldberg [36]. L'idée de base de l'algorithme génétique est d'évoluer une population d'individus (solutions) selon des règles bien précises. Traditionnellement, une solution de la population peut être représentée par un codage binaire qu'on appelle *chromosome*. L'utilisation d'un codage sous forme d'une suite d'entiers s'avère plus efficace pour les problèmes d'optimisation combinatoires [16].

Dans le contexte de l'analogie avec l'évolution des espèces, associe à chaque solution

pouvant être considérée par l'algorithme, une fonction de performance (fitness solution) qui devrait correspondre à sa capacité à survivre et à se reproduire. Celle-ci correspond souvent à la fonction objectif du problème que l'on tente de résoudre ou y est reliée de près. Dans la majorité de problèmes d'optimisation, la solution recherchée doit satisfaire certaines contraintes. On utilise souvent une fonction de pénalité pour diminuer la fonction de performance des individus qui ne respectent pas ces contraintes.

L'étape initiale d'un algorithme génétique consiste à engendrer aléatoirement une population initiale contenant un certain nombre d'individus dont on évalue ensuite la fonction de performance. On applique ensuite successivement les quatre opérateurs suivants pendant un certain nombre d'itérations :

1. **Sélection :** L'opérateur de sélection choisit sur une base aléatoire deux parents qui vont se "reproduire" pour "donner naissance" (créer) un nouvel individu. Le choix peut être fait de différentes façons : les parents peuvent tirés au hasard de façon équiprobable ou en faisant intervenir leur fonction de performance (ce que l'on fait habituellement).
2. **Croisement :** Cet opérateur est un mécanisme de reproduction qui combine des éléments du patrimoine génétique des parents sélectionnés à l'étape précédente pour constituer le patrimoine génétique de l'enfant. Cette opération de croisement est souvent réalisée en coupant en un ou deux points choisis aléatoirement les chromosomes des parents, puis en échangeant de façon appropriée les composantes des deux solutions. Ainsi, dans le croisement avec deux points de coupure, la solution enfant est identique au premier ou au second parent, sauf pour les éléments situés entre les deux points de coupure qui sont recopiés de l'autre parent. Un exemple de cet opérateur avec coupures aux 3ème et 6ème éléments est illustré dans le Tableau 1.1. Généralement, pour les algorithmes génétiques deux solutions enfants sont générées avec un croisement à deux points en inversant les rôles des solutions parents. Dans notre implémentation nous avons choisi de n'utiliser qu'une seule solution enfant dans le but de diversifier notre recherche.
3. **Mutation :** L'opérateur de mutation modifie légèrement chaque enfant obtenu dans l'étape précédente selon avec une faible probabilité dans le but d'éviter

que tous les individus de la population deviennent trop semblables, ce qui ferait stagner la recherche. Cet opérateur a donc une fonction de diversification de la recherche.

4. **Remplacement (culling) :** Cet opérateur permet de produire la nouvelle population en éliminant les individus les moins performants dans la population pour ainsi favoriser les plus prometteurs.

Finalement, on doit définir un critère d'arrêt à la recherche. On peut choisir, par exemple, une limite sur le nombre maximum d'itérations ("générations") après lequel la recherche doit s'arrêter ou des critères liés à des mesures prises sur la population (par exemple, relativement à la similarité des solutions).

parent 1	**3**	**6**	**7**	**0**	**0**	**2**	**1**	**7**	**4**	**3**	**5**	**7**
parent 2	1	4	1	3	2	2	1	9	0	0	2	1
Enfant	**3**	**6**	1	3	2	2	**1**	**7**	**4**	**3**	**5**	**7**

TAB. 1.1: Un exemple de croisement avec coupures aux 3ème et 6ème éléments.

1.7.3 Partition de l'espace de recherche

En général, la structure du système série-parallèle que nous étudions est donnée sous forme d'une suite d'entiers $\mathbf{X} = (X_l)$ tel que $1 \leq l \leq M$. Dans cette sous-section nous proposons l'idée de partition de l'espace de recherche en s'appuyant sur les définitions suivantes :

Définition 2

L'adresse de $\mathbf{X}$ est définie par : $address(\mathbf{X}) = \sum_{l=1}^{M} X_l$.

Cette adresse peut représenter par exemple le nombre total des composantes dans le système. Pour plus de détail sur cette adresse, nous référons le lecteur aux chapitres 2, 3 et 4.

Définition 3

Un sous-espace de recherche qui contient des solutions de même adresse r est un sous-espace d'adresse r (noté par S_r).

Remarque 1

D'après les définitions précédentes, la borne inférieure et la borne supérieure de r sont respectivement, s, le nombre de sous-systèmes (dans ce cas, chaque sous-système contient une et une seule composante) et $N = \sum_{l=1}^{M} \max(X_l)$ (chaque sous-système contient toutes les composantes disponibles dans le marché).

Proposition 1

$(S_r)_{s \leq r \leq N}$ est une partition de l'espace de recherche S.

1.7.4 Particularités de recherche avec tabous appliquée à un sous-espace

Dans la section précédente, l'espace global de recherche, qui correspond à l'ensemble des structures possibles du système série-parallèle, est divisé en plusieurs sous-espaces de recherche disjoints. Pour appliquer l'algorithme de recherche avec tabous à un sous-espace quelconque, on doit définir les voisinages et les mouvements de façon à ce que la recherche demeure toujours dans le même sous-espace après une transformation locale de la solution actuelle. À chaque itération de l'algorithme de RT, les transformations (ou les mouvements), qui peuvent être appliquées à une solution X, définissent un ensemble (voisinage) de solutions voisines de X donné par : $N(X)$= {*structures série-parallèles obtenues par l'application d'un seul mouvement*}. Un mouvement appliqué à la solution actuelle consiste par exemple à ajouter 1 à une composante quelconque de la solution X et à soustraire 1 à une autre composante de X. Donc, l'adresse de cette solution reste la même après cette transformation. Pour illustrer ce mouvement, on considère la

structure suivante : $X = (2, 3, 5, 9)$. L'adresse de cette solution est 19. on suppose que $\max(X_l) = 10\ \forall l,\ 1 \leq l \leq M$. Le voisinage de X est donnée par : $N(X)$= {(3, 2, 5, 9), (3, 3, 4, 9), (3, 3, 5, 8), (1, 4, 5, 9) ,(2, 4, 4, 9), (2, 4, 5, 8), (1, 3, 6, 9), (2, 2, 6, 9), (2, 3, 6, 8), (1, 3, 5, 10), (2, 2, 5, 10), (2, 3, 4, 10)}. La taille de ces voisinages est donc de l'ordre de $O(M^2)$, où M est le nombre de composantes des vecteurs représentant les solutions. Pour la plupart des instances, M^2 est autour de 10^3 pour les trois premiers problèmes et autour de 10^4 pour le quatrième problème. Pour plus de détails sur la recherche avec tabous utilisée dans nos méthodes de résolution, nous référons le lecteur :

- au chapitre 2 pour le problème d'allocation de la redondance dans un système multi-états série-parallèle, dans le cas homogène ;
- au chapitre 3 pour le problème d'allocation de la redondance dans un système multi-états série-parallèle, dans le cas non homogène ;
- au chapitre 4 pour les problèmes d'allocation de la redondance des systèmes série-parallèles binaires, et de planification optimale des extensions pour des systèmes multi-états.

1.7.5 Hybridation de l'algorithme génétique et recherche avec tabous

L'hybridation de l'algorithme génétique et de la recherche locale est reconnue comme étant une approche efficace pour résoudre les problèmes combinatoires difficiles depuis plusieurs années. Cette hybridation combine deux avantages complémentaires : L'AG diversifie la recherche en l'orientant vers des régions non encore explorées et qui semblent les plus intéressantes, tandis que le rôle de la RT est d'intensifier la recherche de l'optimum global dans ces régions. Concrètement, notre algorithme est un AG où l'opérateur de mutation est remplacé par la recherche avec tabous. Les principales caractéristiques de notre algorithme peuvent être résumées comme suit (voir également Fig. 1.6) :

1. **Représentation de la solution :** Celle-ci est codée sous forme d'une suite d'entiers.
2. **Population initiale :** Elle est générée aléatoirement.
3. **Sélection :** Dans notre opérateur, les chances de sélections sont égales pour tous

les individus (les solutions).

4. **Croisement :** Nous utilisons l'opérateur avec deux points de coupure décrit dans la section 1.7.2.
5. **Intensification :** Cette phase remplace la phase de mutation des algorithmes génétiques classiques. On applique ici la recherche avec tabous au sous-espace engendré par l'enfant créé à l'étape précédente en partant de cette solution comme point de départ. La solution obtenue par la RT devient le nouvel enfant.
6. **Remplacement :** Après avoir évalué la nouvelle solution obtenue de la RT, on compare sa valeur avec celle de la pire solution présente dans la population. Si la nouvelle solution est meilleure, elle remplace l'autre dans la population ; sinon, elle est tout simplement rejetée.
7. **Critère d'arrêt :** On fixe a priori le nombre maximum d'itérations après lequel la recherche doit s'arrêter.

En se basant sur l'idée de partitionnement de l'espace de recherche, on veut éviter l'application de l'algorithme de recherche avec tabous plus d'une fois au même sous espace de recherche. Une liste L (initialement vide) est donc utilisée pour stocker les adresses des sous-espaces déjà visités. L'opérateur d'intensification ne sera pas appliqué si l'adresse de l'enfant se trouve dans la liste L.

1.7.6 Calibrage des paramètres

Le réglage des paramètres des algorithmes d'optimisation proposés pour résoudre nos problèmes reste encore une question ouverte et une tâche très compliquée. Un des avantages de nos approches est lié au fait qu'elles utilisent peu de paramètres. Le paramètre de pénalité ajouté dans la fonction objective est le seul qui est difficile à contrôler. Les autres paramètres utilisés dans nos algorithmes sont le nombre de solutions générées aléatoirement dans la population, le nombre d'itérations pour l'algorithme de recherche avec tabous, la taille maximale de la liste taboue et le nombre maximal de cycles pour l'algorithme génétique. Pour chaque problème, les valeurs de ces paramètres ont été fixées après avoir effectué des tests sur un ensemble réduit d'ins-

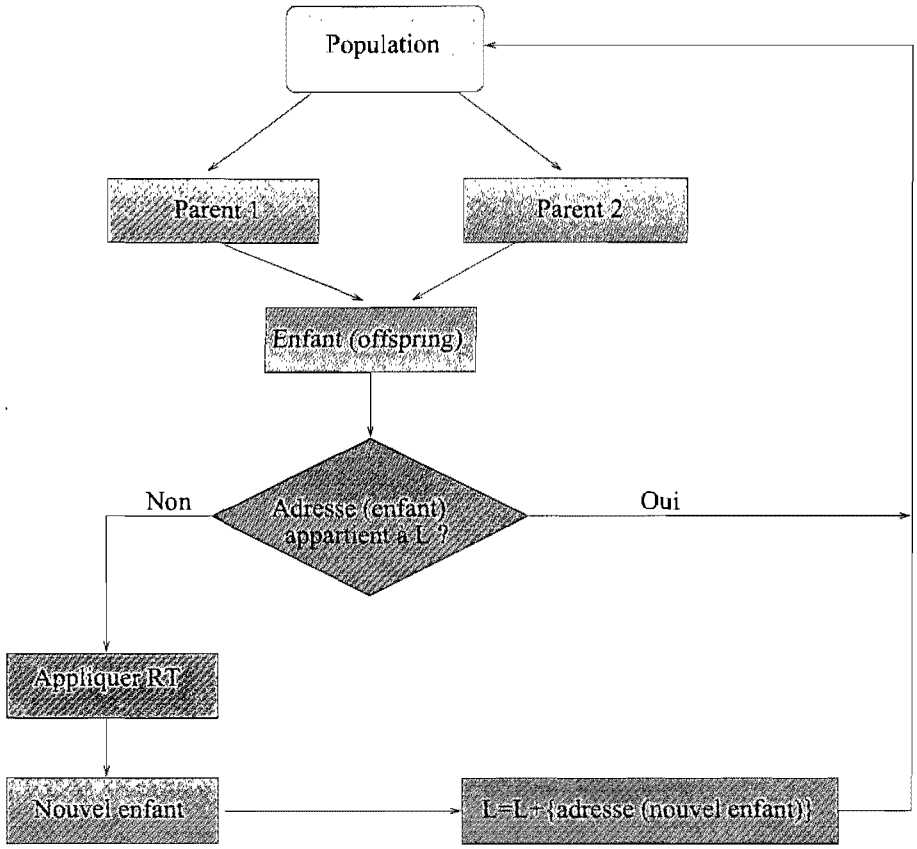


FIG. 1.6 – Hybridation de l'algorithme génétique et la recherche avec tabous

tances générées aléatoirement. Pour chaque paramètre, nous avons considéré un certain nombre de valeurs qui paraissaient raisonnables et nous avons résolu les instances retenues avec chacune de ses valeurs. Les valeurs considérées varient pour chacun des paramètres et vont dévuter en détail dans chacun des articles en cause.

1.8 Plan de la thèse

Cette thèse, rédigée selon le principe d'insertion d'articles, est composée de trois articles en anglais qui peuvent être lus indépendamment.

Le chapitre 2 étudie la première version du problème d'allocation de la redondance, pour un système multi-états série-parallèle (cas homogène). Nous proposons un algo-

rithme de recherche tabou appliqué à plusieurs sous-espaces de recherche disjoints pour résoudre ce problème. L'article correspondant à ce chapitre a été publié en 2008, dans le volume 93 de la revue *Reliability Engineering and System Safety.*

Le chapitre 3 traite le problème d'allocation de la redondance dans un système multi-états série-parallèle, dans le cas non homogène. Nous améliorons le premier algorithme en ajoutant l'algorithme génétique pour sélectionner les sous-espaces de recherche. Il s'agit spécifiquement d'une hybridation de deux métaheuristiques, l'algorithme de recherche tabou et l'algorithme génétique. L'article correspondant à ce chapitre a été soumis pour publication dans *Journal of Heuristics* en Janvier 2009.

Le chapitre 4 est consacré à la résolution du problème d'allocation de la redondance des systèmes série-parallèles binaires, et du problème de la planification optimale des extensions pour des SME. L'article correspondant à ce chapitre a été accepté en 2009 pour publication dans le journal *Computers and Operations Research.*

Le chapitre 5 rassemble les travaux de cette thèse en résumant les différentes idées apportées. Certaines améliorations possibles sont proposées et quelques idées de recherche pour de futurs travaux sont émises.

1.9 Conclusion

Dans ce chapitre nous avons introduit la problématique générale et les objectifs de la thèse. Ensuite, nous avons présenté une revue des principaux travaux publiées et un résumé de la méthodologie. Les trois prochains chapitres présentent trois contributions originales de la thèse, au domaine de la conception et la planification optimales des systèmes séries-parallèles.

Chapter 2

Tabu search for the redundancy allocation problem of homogenous series-parallel multi-state systems

Résumé. La plupart des travaux existants sur la théorie de la fiabilité ont développé la fiabilité binaire qui permet d'étudier les systèmes à deux états (fonctionnement parfait ou bien une panne totale). Une modélisation de type multi-états est plus complexe mais elle est plus riche en information. Dans ce chapitre, nous présentons une approche de résolution du problème d'allocation de la redondance, pour un système multi-états série-parallèle *homogène*. En plus de l'originalité de la méthodologie proposée dans cet article, notons que la modélisation du système étudié selon la théorie de la fiabilité des systèmes multi-états revêt un intérêt particulier dans cette thèse. En effet, il existe plusieurs situations en pratique dans lesquelles un système de production doit être considéré comme un SME. Si les systèmes sont réparables alors la disponibilité est la mesure la plus appropriée pour décrire leur endurance. La disponibilité du SME sera mesurée par la probabilité que le système satisfasse la demande (un exemple de la demande est

présenté dans la sous-section (1.2.2) de premier chapitre). Un système multi-états est caractérisé par son évolution dans un espace d'états où l'ensemble des états possibles est considéré fini. Chaque état dépend de la relation entre la performance totale du système et la demande. En général, la demande est une variable aléatoire. Dans cette thèse, nous supposons que, la distribution de la demande est prédite sous la forme d'une courbe de demande cumulative par morceaux.

Des composantes identiques sont incluses en parallèle dans chaque sous-système afin de parvenir à un niveau souhaitable de disponibilité. Les composantes du système sont caractérisées par leur coût, leur performance et leur disponibilité. L'objectif est de minimiser la somme des coûts d'investissement, tout en satisfaisant la contrainte de disponibilité du système. Le problème d'allocation de la redondance dans les SME séries-parallèles a été introduit pour la première fois par Ushakov dans [80], où la fonction génératrice universelle a été utilisée pour calculer la disponibilité. Suivant ces travaux, les chercheurs ont utilisé des métaheuristiques pour trouver une configuration optimale pour un SME. Notre proposition consiste à partitionner l'espace global de recherche et à appliquer l'algorithme de recherche tabou sur plusieurs sous-espaces disjoints. Les résultats numériques obtenus pour trois benchmarks proposés dans la littérature et un quatrième benchmark de plus grande taille généré aléatoirement, montrent que l'algorithme proposé présente des avantages, tant au niveau du temps d'exécution qu'au niveau de la qualité des solutions obtenues.

Tabu search for the redundancy allocation problem of homogenous series-parallel multi-state systems

Mohamed OUZINEB
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT),
Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Mustapha NOURELFATH
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT),
Département de génie mécanique,
Université Laval, Local 3344, Pavillon Adrien-Pouliot Sainte-Foy,
Québec G1K 7P4, Canada.

Michel GENDREAU
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT),
Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Article a été publié dans le volume 93 de la revue *Reliability Engineering and System Safety*

March 2008

Abstract. This paper develops an efficient tabu search heuristic to solve the redundancy allocation problem for multi-state series-parallel systems. The system has a range of performance levels from perfect functioning to complete failure. Identical redundant elements are included in order to achieve a desirable level of availability. The elements of the system are characterized by their cost, performance and availability. These elements are chosen from a list of products available in the market. System availability is defined as the ability to satisfy consumer demand which is represented as a piecewise cumulative load curve. A universal generating function technique is applied to evaluate system availability. The proposed tabu search heuristic determines the minimal cost system configuration under availability constraints. An originality of our approach is that it proceeds by dividing the search space into a set of disjoint subsets, and then by applying tabu search to each subset. The design problem, solved in this study, has been previously analyzed using genetic algorithms. Numerical results for the test problems from previous research are reported, and larger test problems are randomly generated. Comparisons show that the proposed tabu search outperforms genetic algorithm solutions, both in terms of the solution quality and the execution time.

Keywords. Redundancy allocation; Multi-state systems, Series-parallel systems, Meta-heuristics, Tabu search; Genetic algorithm, Universal generating function

Abbreviations & Acronyms

RAP	Redundancy Allocation Problem
MSS	Multi-State System
UMGF	Universal Moment Generating Function
TS	Tabu Search
GA	Genetic Algorithm
MPCI	Maximal Possible Cost Improvement
MPTI	Maximal Possible Time Improvement

Notation

s	number of series MSS subsystems or components
i	index for component $Comp_i$, $i \in \{1, 2, ..., s\}$
$\max(J_i)$	maximum number of versions available in the market for element belonging to $Comp_i$
J_i	a number which identifies version (version number or index) of element belonging to $Comp_i$, $J_i \in \{1, 2, ..., \max(J_i)\}$
J	a vector $(J_i)_{1 \leq i \leq s}$ which specifies version numbers for each component in the system
X_i	number of elements connected in parallel in $Comp_i$
$\max(X_i)$	maximum X_i allowed (i.e. the upper bound of X_i)
X	a vector $(X_i)_{1 \leq i \leq s}$ which specifies the numbers of elements connected in parallel for each component in the system, $X_i \in \{1, 2, ..., \max(X_i)\}$
Y	a vector $(Y_l)_{1 \leq l \leq 2s}$ of dimension $2s$ which defines the entire system structure. For $1 \leq l \leq s$, the components of the vector Y are those of X. For $s+1 \leq l \leq 2s$, the components of the vector Y are those of J. This vector is denoted by $Y = (X, J)$
$C(Y)$	total cost of series-parallel MSS, $C(Y) = C(X, J)$
$A(Y)$	stationary availability index of the overall multi-state series-parallel system, A(Y)=A(X,J)

A_0	a specified minimum required level of system availability index
j	index for element version ($j = J_i$)
A_{ij}	binary-state availability of element of version j belonging to $Comp_i$
C_{ij}	cost of each element of version j in component $Comp_i$
W_{ij}	nominal performance level of element of version j in component $Comp_i$
K	number of partitioned intervals
k	index for partitioned intervals
T	MSS operation period
T_k	a partitioned interval in T, $T = \sum_{k=1}^{K} T_k$
$\mathbf{T}$	vector $(T_k)_{1 \le k \le K}$
W_k^0	required MSS performance level for T_k
$\mathbf{W}^0$	vector $(W_k^0)_{1 \le k \le K}$
R_{ms}	MSS reliability
A_{ms}	MSS stationary availability
W_S	total capacity of the system
$y(t)$	MSS state at time t, $y(t) \in \{1, 2, ..., M\}$, 1 is the worst state and M is the best state
m	number or index of MSS state, $m \in \{1, 2, ..., M\}$, 1 is the worst state and M is the best state
W_m	MSS steady-state performance level associated with m
$W(t)$	output performance level of the MSS at time t, $W(t) \in \{W_1, ..., W_M\}$
p_m	$\lim_{t \to \infty}[Pr(W(t) = W_m)]$
$mngi$	maximum number of global iterations after the best known solution
$mnli$	maximum number of local iterations after the best known solution
q	amplification parameter in the penalized objective function
P_m	mutation probability
N_s	number of randomly-constructed solutions in the initial population of GA
N_c	number of genetic cycles
N_{rep}	number of reproduction-selection procedures per genetic cycle
CPU	the running time given in second (time of execution)

2.1 Introduction

The RAP is a complex combinatorial optimization problem, which is very important in many industrial applications. These applications include power systems, electronic systems, telecommunications systems and manufacturing production systems. In the formulation of the RAP of a series-parallel structure, the system consists of components in series, and for each component, multiple element choices are used in parallel. Thus, the design goal is to select the optimal combination of elements and redundancy levels to meet system level availability or reliability constraints while minimizing total investment-cost. The RAP is NP-hard [14]. Most existing works on RAP use traditional binary-state reliability. Binary-state reliability modeling assumes that a system and its elements may experience only two possible states: good and failed. The RAP for binary-state series-parallel systems has been studied in many different forms, and by considering numerous approaches and techniques, as summarized in [41, 42, 77].

The multi-state version of the problem is more recent and has not been sufficiently studied. Multi-state reliability modeling considers that a system and its elements may experience a range of performance levels from perfect functioning to complete failure. For reparable MSS, availability is defined as the ability to satisfy consumer demand which is represented as a piecewise cumulative load curve. Others papers on MSS include [2, 10, 60, 66]. A good and extensive review of MSS literature can be found for example in [46, 49, 55, 82]. The RAP for series-parallel MSS was first introduced in [81], where the universal generating function method was used for the reliability calculation [80].

Following these works, genetic algorithms were used in [50, 51, 56] for finding the minimal cost configuration of a series-parallel MSS under reliability or availability constraints. The only others existing solution techniques of this problem are ant colony optimization [68] and a heuristic algorithm in [73]. However, the best-published results in terms of the solution quality and the execution time have been provided by genetic algorithms in [50, 51, 56].

The purpose of the present paper is to develop an efficient tabu search to solve the RAP for series-parallel MSS. Elements of same type are used to provide redundancy. The results of the newly developed tabu-search heuristic are compared with the best-published results from the literature, i.e., those of genetic algorithms. Tabu search is a meta-heuristic method originally proposed in [29]. Like other local search methods, tabu search is an iterative procedure that, beginning with an initial feasible solution, progressively improves it by applying a series of local modifications or moves within a neighborhood. The basic principle of tabu search is to pursue the search whenever a local optimum is encountered by allowing non-improving moves; cycling back to previously visited solutions is prevented by the use of memories, called tabu lists, that record the recent history of the search. Over the last twenty years, hundreds of papers presenting applications of tabu search to various combinatorial problems have appeared in the operations research literature (see, for example, [22, 23, 25, 26, 33, 34]). In several cases, the methods described provide solutions very close to optimality and are among the most effective, if not the best, to tackle the difficult problems at hand. These successes have made tabu search extremely popular among those interested in finding good solutions to the large combinatorial problems encountered in many practical settings. Because of the large search space size of the RAP for series-parallel MSS, and given its natural neighborhood structure, it is a good candidate for tabu search.

The remainder of the paper is organized as follows. Section 2 provides a description of the redundancy allocation problem for series-parallel multi-state systems. The MSS availability estimation method is presented in Section 3. The solution approach and the TS heuristic to solve the problem are introduced in Section 4. In Section 5, we present the test problems and the numerical results. Concluding remarks are made in Section 6.

2.2 The redundancy allocation problem for series-parallel multi-state systems

2.2.1 General description of the problem and assumptions

The series-parallel MSS consists of a number of components connected in series, such that each component can contain several elements connected in parallel. Failed elements are repaired and the elements availabilities are known. The MSS is reparable and its availability is defined as its ability to satisfy consumer demand which is represented as a piecewise cumulative load curve. For each component there are various element versions, which are proposed by the suppliers on the market. These elements are characterized according to their version by their cost, availability and performance. The capacity or productivity of elements is the performance measure. Redundancy allows availability improvement, but increases the total cost. The objective is to design the series-parallel system so that the total cost is minimized, subject to a multi-state availability constraint. It is considered that once an element selection is made, only the same element type can be used to provide redundancy. That is, for each component, one has to select one element type to be used and to determine the number of redundant elements. There are no analytical functions that express the dependence of the component cost on its nominal performance-level and reliability or availability. In addition the following assumptions are considered:

1. Failure of individual elements are independent.
2. There exists a steady-state distribution of MSS state probabilities.
3. Mixing of components is not allowed.
4. Each element may experience only two possible states: good and failed.

2.2.2 Mathematical formulation of the problem

Given a system structure defined as a vector $Y = (X, J)$ of dimension $2s$ ($Y = (Y_l)_{1 \leq l \leq 2s)}$, the total cost can be calculated by the sum of the costs of the chosen elements. The cost of component $Comp_i$ is given by $X_i C_{iJ_i}$. Thus, the total cost (objective function) is given by:

$$C(X, J) = \sum_{i=1}^{s} X_i C_{iJ_i}. \tag{2.1}$$

If one has to take into account price discounting, the element cost should be considered as a function of the number of elements purchased simultaneously. In this case, the total cost is a function of X_i:

$$C(X, J) = \sum_{i=1}^{s} X_i C_{iJ_i}(X_i). \tag{2.2}$$

The problem can be formulated as follows: find the minimal cost system structure (X, J) that meets or exceeds the required availability A_0. That is,

$$\text{minimize } C(X, J) = \sum_{i=1}^{s} X_i C_{iJ_i}(X_i) \tag{2.3}$$

subject to

$$A(X, J) \geq A_0, \tag{2.4}$$

$$X_i \in \{0, 1, \ldots \max(X_i)\} \qquad \forall i, \quad 1 \leq i \leq s, \tag{2.5}$$

$$J_i \in \{0, 1, \ldots \max(J_i)\} \qquad \forall i, \quad 1 \leq i \leq s. \tag{2.6}$$

Constraint (2.5) specifies that, for each component $Comp_i$, the number of elements connected in parallel is an integer which cannot be higher than a pre-selected maximum number $\max(X_i)$. Constraint (2.6) specifies that, for each component $Comp_i$, versions are identified by integers from 1 to a maximum number of versions available in the market. Given the solution structure ($Y = (X, J)$), the identical elements constraint is verified automatically.

2.2.3 Availability of repairable multi-state systems

The MSS reliability is [86]:

$$R_{ms}(t, W^0) = Pr(W(t) \geq W^0). \tag{2.7}$$

It is assumed that there exists a stationary distribution of failures (steady-state). This hypothesis is acceptable and provided by excluding some phenomena that could affect the distribution of elements, or to consider a long production periods in order to be sure that the system can be in a steady-state. For repairable MSS, a multi-state stationary availability A_{ms} is used as $Pr(W(t) \geq W^0)$ after enough time has passed for this probability to become constant [51]. In the steady-state the distribution of states probabilities is given by equation (2.8), while the multi-state stationary availability is formulated by equation (2.9):

$$p_m = \lim_{t \to \infty} [Pr(W(t) = W_m)], \tag{2.8}$$

$$A_{ms}(W^0) = \sum_{W_m \geq W^0} p_m. \tag{2.9}$$

Let us consider an operation period T that is divided into K intervals. Each interval has a duration T_k and a required demand level $(k = 1, ..., K)$. In this case, the MSS availability index is:

$$A(Y) = \frac{1}{\sum_{k=1}^{K} T_k} \sum_{k=1}^{K} A_{ms}(W_k^0) T_k. \tag{2.10}$$

If we denote by $Pr(W_S \geq W_k^0)$ the probability that the total capacity of the system (W_S) is not lower than a specific demand level W_k^0 , equation (2.10) can be written as follows:

$$A(Y) = \frac{1}{\sum_{k=1}^{K} T_k} \sum_{k=1}^{K} Pr(W_S \geq W_k^0) T_k. \tag{2.11}$$

In power engineering, for example, $A(Y)$ is often related to the loss of load probability (LOLP) index [9, 56] as LOLP $= 1 - A(Y)$. LOLP is understood as the probability that the system cannot supply a given demand load.

2.3 MSS availability estimation method

To solve the combinatorial optimization problem formulated in subsection 2.2 by equations (2.3)-(2.6), it is necessary to do numerous estimations of MSS cost and availability. It is therefore crucial to have an effective and fast procedure to evaluate $A(X, J)$. For this evaluation, we use the universal z-transform [80], which has been proven to be very effective for high-dimension combinatorial optimization problems [49]. This mathematical technique is also called, in the literature, universal moment generating function (UMGF) or simply U-function or U-transform. The U-function extends the widely known ordinary moment generating function [75].

2.3.1 Definitions and properties of the U-function

Definition 1

The U-function of a discrete random variable W_S is defined as a polynomial:

$$U(z) = \sum_{m=1}^{M} p_m z^{W_m}, \tag{2.12}$$

where the variable W_S has M possible values and p_m is the probability that is W_S equal to W_m (given by equation (2.8)).

□

Definition 2

If the discrete random variable W_S is the MSS stationary output performance, then the availability $A_{ms}(W_k^0)$ is given by the probability:

$$P[W_S \geq W_k^0] = \Phi(U(z)z^{-W_k^0}), \tag{2.13}$$

where Φ is a distributive operator defined by:

$$\Phi(pz^{\alpha}) = p1_{[\alpha \geq 0]}, \tag{2.14}$$

$$1_{[True]} = 1, \tag{2.15}$$

$$1_{[False]} = 0, \tag{2.16}$$

$$\Phi(\sum_{m=1}^{M} p_m z^{\alpha_m}) = \sum_{m=1}^{M} \Phi(p_m z^{\alpha_m}). \tag{2.17}$$

□

It can be shown that Eqs. (2.12)-(2.17) meet condition $P[W_S \geq W_k^0] = \sum_{W_m \geq W_k^0} p_m$. In fact:

$$\begin{aligned} P[W_S \geq W_k^0] &= \Phi(U(z)z^{-W_k^0}) \\ &= \Phi(\sum_{m=1}^{M} p_m z^{W_m - W_k^0}) \\ &= \sum_{m=1}^{M} \Phi(p_m z^{W_m - W_k^0}) \\ &= \sum_{m=1}^{M} p_m 1_{[W_m - W_k^0 \geq 0]} \\ &= \sum_{W_m \geq W_k^0} p_m. \end{aligned}$$

By using the operator Φ, the coefficients of polynomial $U(z)$ are summed for every term with $W_m \geq W_k^0$, and the probability that W_S is not less than some arbitrary value W_k^0 is systematically obtained. As an example, consider single elements with total failures, and each element j has nominal performance W_j and availability A_j. Then, $Pr[W_S = W_j] = A_j$ and $Pr[W_S = 0] = 1 - A_j$. The UMGF of such an element has only two terms and can be defined as:

$$U_j(z) = (1 - A_j)z^0 + A_j z^{W_j} = (1 - A_j) + A_j z^{W_j}. \tag{2.18}$$

The essential property of the U-transform allows the total UMGF for a system of elements connected in parallel or in series to be obtained using simple algebraic operations over the individual U-functions of elements. To perform such operations, any used operator ψ has to satisfy the following Ushakov's four properties [80]:

1. $\psi(pz^a) = pz^a$.
2. $\psi(p_1 z^{a_1}, p_2 z^{a_2}) = p_1 p_2 z^{f(a_1,a_2)}$, where $f(a_1, a_2)$ is defined according to the physical nature of the MSS performance and the interactions between MSS elements.
3. $\psi(U_1(z), ..., U_k(z), U_{k+1}(z), ...U_n(z)) = \psi(\psi(U_1(z), ..., U_k(z)), \psi(U_{k+1}(z), ..., U_(z)))$ for any k.
4. $\psi(U_1(z), ..., U_k(z), U_{k+1}(z), ...U_n(z)) = \psi(U_1(z), ..., U_{k+1}(z), U_k(z), ...U_n(z))$ for any k.

2.3.2 Series-parallel MSS availability evaluation using U-functions

To evaluate the availability of a series-parallel MSS, two basic composition operators are introduced. These operators determine the polynomial $U(z)$ for a group of elements.

Parallel elements

When the performance measure is related to the system productivity or capacity, the total performance of the parallel system is the sum of performances of all its elements. The function to be used for operator ψ in property 2 above is therefore: $f(a_1, a_2) = a_1 + a_2$.

The U-function of MSS component $Comp_i$ containing X_i parallel elements can be calculated by using the Γ operator:

$$U_p(z) = \Gamma(U_1(z), U_2(z), ..., U_{X_i}(z)), \text{ where } f(a_1, a_2, ..., a_n) = \sum_{i=1}^{X_i} a_i. \qquad (2.19)$$

Therefore, for a pair of elements connected in parallel:

$$\Gamma(U_1(z), U_2(z)) = \Gamma(\sum_{i=1}^{n} P_i z^{a_i}, \sum_{j=1}^{m} Q_j z^{b_j}) = \sum_{i=1}^{n}\sum_{j=1}^{m} P_i Q_j z^{a_i+b_j}. \quad (2.20)$$

Parameters a_i and b_j are physically interpreted as the respective performances of the two elements. n and m are numbers of possible performance levels for these elements. P_i and Q_j are steady-state probabilities of possible performance levels for elements. One can see that the Γ operator is simply a product of the individual U-functions. Thus, the component UMGF is:

$$U_p(z) = \prod_{e=1}^{X_i} U_e(z). \quad (2.21)$$

Series elements

When the elements are connected in series, the element with the least performance becomes the bottleneck of the system. Thus, this element may define the total system productivity. To calculate the U-function for system containing s elements connected in series, the operator η should be used:

$$U_s(z) = \eta(U_1(z), U_2(z), ..., U_s(z)), \text{ where } f(a_1, a_2, ..., a_s) = \min\{a_1, a_2, ..., a_s\}. \quad (2.22)$$

Therefore, for a pair of elements connected in series:

$$\eta(U_1(z), U_2(z)) = \eta(\sum_{i=1}^{n} P_i z^{a_i}, \sum_{j=1}^{m} Q_j z^{b_j}) = \sum_{i=1}^{n}\sum_{j=1}^{m} P_i Q_j z^{\min\{a_i,b_j\}}. \quad (2.23)$$

Series-parallel systems

The UMGF of the entire series-parallel system (containing s series components) may be obtained by applying composition operators consecutively. Assuming that each element has 2 states (nominal performance or total failure), and given the individual UMGF of elements defined in equation (2.18), the U-function of $Comp_i$ containing X_i parallel elements is calculated as follows:

$$U_i(z) = \Gamma(U_1(z), U_2(z), ..., U_{X_i}(z)) = \prod_{k=1}^{X_i} [(1 - A_{ij}) + A_{ij} z^{W_{ij}}]. \quad (2.24)$$

Under the assumption that only identical elements are used to provide redundancy, the U-function of $Comp_i$ can be represented as [51]:

$$U_i(z) = [(1 - A_{ij}) + A_{ij} z^{W_{ij}}]^{X_i} = \sum_{k=0}^{X_i} \alpha_{ik} z^{kW_{ij}}, \tag{2.25}$$

$$\alpha_{ik} = \text{binm}(k, A_{ij}, X_i) = [\frac{X_i!}{k!(X_i - k)!}] A_{ij}^k (1 - A_{ij})^{X_i - k}. \tag{2.26}$$

The UMGF of the entire system containing s components connected in series is:

$$\begin{aligned} U(z) &= \eta(U_1(z), U_2(z), ..., U_s(z)) \\ &= \eta(\sum_{k=0}^{X_1} \alpha_{1k} z^{kW_{1j}}, \sum_{k=0}^{X_2} \alpha_{2k} z^{kW_{2j}}, ..., \sum_{k=0}^{X_s} \alpha_{sk} z^{kW_{sj}}) = \sum_{m=0}^{M} \delta_m z^{W_m}, \end{aligned} \tag{2.27}$$

where δ_m and W_m are real numbers obtained by using Eq. (2.23).

To evaluate the probability that the entire system provides a performance level exceeding W_k^0, the operator Φ is applied to Eq. (2.27) as follows:

$$A_{ms}(W_k^0) = Pr[W_S \geq W_k^0] = \Phi(U(z) z^{-W_k^0}) = \sum_{W_m \geq W_k^0} \delta_m. \tag{2.28}$$

Finally, the total availability $A(Y)$ for all the demand levels is:

$$A(Y) = \frac{1}{\sum_{k=1}^{K} T_k} \sum_{k=1}^{K} \sum_{W_m \geq W_k^0} \delta_m T_k. \tag{2.29}$$

2.4 The tabu search approach

Tabu search is an iterative procedure, where at each iteration we move from a current solution to a new solution in a neighborhood. Choosing a search space and a neighborhood structure is by far the most critical step in the design of any tabu search heuristic. It is at this step that one must make the best use of the understanding and knowledge

he/she has of the problem at hand [25]. In the problem studied in this paper, the search space is the space of all possible series-parallel structures considered (visited solutions) during the search. A particularity of the proposed approach is that it proceeds in two steps: (i) dividing the search space into a set of disjoint subsets; (ii) applying TS to subspaces.

2.4.1 Partitioning the search space

Definition 3

Given a system structure as a vector $Y = (X, J)$ of dimension $2s$ ($Y = (Y_l)_{1 \leq l \leq 2s}$), the address of Y is defined by:
$address(Y) = \sum_{l=1}^{2s} Y_l$.

□

Example 1

Let consider a system of 2 components in series. Component 1 contains 2 elements of version 5 in parallel, and component 2 contains 3 elements of version 9 in parallel. From definition 3, the address of Y is simply obtained by summing the total number of elements (used in the system) with the sum of version numbers. that is, $address(Y) = (2 + 3) + (5 + 9) = 19$.

Definition 4

A search subspace of address r, denoted by S_r is defined as the set of series-parallel structures (solutions) which have the same address, equal to r.

□

It follows from the above definitions that the lower bound of r is $2s$ and its upper bound is given by $N = \sum_{i=1}^{s} \max(X_i) + \sum_{i=1}^{s} \max(J_i)$.

Proposition 1

$(S_r)_{2s \leq r \leq N}$ is a partition of the search space S. This means that:

(1) $S_{r_1} \bigcap S_{r_2} = \emptyset \quad \forall\, r_1 \neq r_2$, and

(2) $S = \bigcup_{r=2s}^{N} S_r$.

□

Proof

(1) Suppose that $\exists\ r_1 \neq r_2$ with $S_{r_1} \bigcap S_{r_2} \neq \emptyset$. Then, there is a solution Y such that $Y \in S_{r_1}$ and $Y \in S_{r_2}$. This means that $address(Y) = r_1$ and $address(Y) = r_2$; this is a contradiction since $r_1 \neq r_2$.

(2) Because, by definition, $S_r \subseteq S \quad \forall r\ (2s \leq r \leq N)$, $\bigcup_{r=2s}^{N} S_r \subseteq S$. To prove the proposition, we need also to prove that $S \subseteq \bigcup_{r=2s}^{N} S_r$.

Suppose that $S \not\subseteq \bigcup_{r=2s}^{N} S_r$. Then, there is a solution $Y \in S$ such that $Y \notin \bigcup_{r=2s}^{N} S_r$; hence $Y \notin S_r \quad \forall r\ (2s \leq r \leq N)$. Consequently:

$$address(Y) < 2s \text{ or } address(Y) > N. \tag{2.30}$$

Or, for any solution vector $Y = (X, J) = (Y_l)_{1 \leq l \leq 2s}$, we have:

(i) $1 \leq Y_l \leq \max(X_l),\ \forall l$ such that $1 \leq l \leq s$; and

(ii) $1 \leq Y_l \leq \max(J_l),\ \forall l$ such that $s+1 \leq l \leq 2s$.

Therefore, $2s \leq \sum_{l=1}^{2s} Y_l \leq \sum_{l=1}^{s} \max(X_l) + \sum_{l=s+1}^{2s} \max(J_l)$. Thus, $2s \leq address(Y) \leq N$ and (2.30) is a contradiction.

□

2.4.2 Applying tabu search to subspaces

In the previous subsection, the search space S has been partitioned into $(N - 2s + 1)$ subspaces $(S_{2s}, S_{2s+1}, ..., S_N)$. To apply TS to a given subspace S_{r_i} $(2s \leq r_i \leq N)$, let define the neighborhood structure. At each iteration of TS, the local transformations (or moves), that can be applied to the current solution Y, define a set of neighboring solutions in S_{r_i} as: Neighborhood (Y) = {Series-parallel structures obtained by applying a single move to Y}.

The move applied to Y consists in changing the number and the version of elements by adding and subtracting one, if possible, for two components of the solution vector $Y = (Y_l)_{1 \leq l \leq 2s}$. In this way, $address(Y)$ does not change after a local transformation of Y and the search process remains in the same subspace. To illustrate this move operator, consider the system described in example 1, where $Y = (2, 3, 5, 9)$ and the address is 19. We also assume that $\max(X_i) \geq 4$, $i = 1, 2$. Each single move can be applied in three different ways:

1. Changing the number of elements in components 1 and 2. The obtained structures are $(1, 4, 5, 9)$ and $(3, 2, 5, 9)$;

2. Changing version numbers in components 1 and 2. The obtained structures are $(2, 3, 4, 10)$ and $(2, 3, 6, 8)$;

3. Changing both number of elements and version numbers in components 1 and 2. The obtained structures are $(1, 3, 6, 9)$, $(1, 3, 5, 10)$, $(2, 2, 6, 9)$ and $(2, 2, 5, 10)$.

That is, Neighborhood (Y) = {$(1, 4, 5, 9)$, $(3, 2, 5, 9)$, $(2, 3, 4, 10)$, $(2, 3, 6, 8)$, $(1, 3, 6, 9)$, $(1, 3, 5, 10)$, $(2, 2, 6, 9)$ and $(2, 2, 5, 10)$ }.

At each iteration, the best solution Y' in a subset $V(Y) \subseteq$ Neighborhood(Y) is selected. $V(Y)$ (referred to as the effective neighborhood) is generated by eliminating the tabu solutions from Neighborhood(Y). Tabus are stored in a short-term memory of the search (tabu list). A previously visited solution is added to the tabu list in order to forbid the repetition of configurations. That is, tabus are used to prevent cycling

when moving away from local optima through non-improving moves. The size of the tabu list (denoted by *length*) is an important parameter regarding the efficiency of the heuristic; but its best value is not easy to be determined. A dynamic length is used ($0 \leq length \leq mnli$), as it is usually found that it is more efficient to use a variable size tabu list.

The termination criterion used in our implementation here is specified in terms of a maximum number of local iterations ($mnli$) without finding an improvement in the best known solution. An important additional feature of our proposed TS is the utilization of a penalty function while allowing infeasible solutions. That is, the penalized objective function to be minimized is:

$$C_p(X, J) = C(X, J) + q \max\{0, A_0 - A(X, J)\}. \tag{2.31}$$

In Eq. (2.31), a weighted penalty for constraint violation is added to the objective function $C(X, J)$. This is done by allowing solutions with availabilities that are smaller than A_0. This penalty function discourages, but allows, the TS algorithm to search into the infeasible boundary region.

The idea of exploring around boundaries is known to be efficient for the RAP of binary state series-parallel systems. For example, the authors of [15] in their use of GA observed that better final feasible solutions could be found by permitting the exploration of the infeasible region, but by penalizing those solutions at the basis of the infeasibility degree. This idea is also used in [41, 54, 62] for solving the RAP of binary state series-parallel systems with other meta-heuristics, and in [61] for reliability optimization of a series system with multiple-choice and budget constraints.

In TS, allowing infeasible solutions is a well-known idea [24, 28, 30, 31, 33]. An interesting way to find correct weights for constraint violations is to use self-adjusting penalties, i.e., weights are adjusted dynamically on the basis of the recent history of the search. Weights are increased if only infeasible solutions were encountered in the last few iterations, and decreased if all recent solutions were feasible; see [24] for further details. Penalty weights can also be modified systematically to drive the search to cross the feasibility boundary of the search space and thus induce diversification. This

technique, known as strategic oscillation, was first introduced in [28] and used since in several successful TS procedures.
In the sequel, the TS algorithm applied to subspaces will be called TS-Sub.

2.4.3 The general algorithm

The general TS algorithm is based on the proposed following steps:

Step 1. The initial solution is chosen such that each component contains only one element of version one.

Step 2. The address is randomly incremented and the best solution is calculated by applying TS-Sub.

Step 3. The termination criterion is checked. It is defined as the maximum number of global iterations ($mngi$) without finding an improvement in the best known solution. If it is reached, the search is concluded and the best obtained solution is recommended. Otherwise, return to step 2.

2.5 Test problems and numerical results

The goal is to find the minimal-cost feasible MSS series-parallel structure that provides a specified measure A_0. For repairable systems, the measure A is the availability [51], and it is evaluated by using the UMGF method described before (with A_{ij} representing availability of element of version j belonging to $Comp_i$). For non repairable systems, the measure A is rather the reliability, and it is evaluated by using the same UMGF method [50, 56]; the only difference being that each A_{ij} represents reliability (instead of availability) of element of version j belonging to $Comp_i$.

All the algorithms were implemented in C^{++}. The numerical tests were completed on an Intel Pentium IV 3000 MHz DEC station 5000/240 with 1024 Mbytes of RAM

running under Linux. All computations use real float point precision without truncating or rounding values. In order to compare with GA results in [50, 51, 56], the system reliability or availability of the final solution is rounded to four digits behind the decimal point.

2.5.1 Test problems

To analyze the solution method introduced in this paper, four design optimization problems (benchmarks) are used. These problems do not allow element mixing. The first three problems were first introduced and solved by GA in [50, 51, 56], while the fourth is a larger new problem which is randomly generated.

Benchmarks data

Problem 1

The problem in [56] consists of four components connected in series, and for each component, from 7 to 11 different elements types are available. The reliability index A_0 has been set to nine different values, from 0.91 to 0.999, to create nine variations of the problem. Table 2.1 presents the problem data used for the nine examples, while Table 2.2 contains the data of piecewise cumulative load demand curve. The interpretation of the data shown in Table 2.2 is that the system needs a supply of demand level W_k^0 during T_k units of time. To take into account possible discounts, the cost of the element is presented as a function of number of elements purchased. This function is defined as follows:

$$C(m) = \begin{cases} C^* & \text{si } m \leq m_1, \\ \gamma_1 C^* & \text{si } m_1 < m \leq m_2, \\ \gamma_2 C^* & \text{si } m_2 < m, \end{cases} \tag{2.32}$$

where: $C^* = C_{ij}$ is the cost per unit and $m_1, m_2, \gamma_1, \gamma_2$ are shown in Table 2.3 for each element type.

Comp i	Element version j	A_{ij}	Cost C_{ij}(mln \$)	Nominal performance W_{ij}(%)
1	1	0.990	1.117	25
	2	0.996	1.310	25
	3	0.995	1.903	35
	4	0.961	1.640	35
	5	0.993	2.122	50
	6	0.957	1.910	50
	7	0.942	1.722	50
	8	0.991	2.591	72
	9	0.951	2.001	72
	10	0.986	3.284	100
	11	0.979	3.095	100
2	1	0.967	4.010	40
	2	0.914	3.450	40
	3	0.960	4.350	55
	4	0.953	4.840	78
	5	0.920	4.210	78
	6	0.950	5.800	93
	7	0.948	6.550	110
3	1	0.967	0.636	25
	2	0.952	0.448	35
	3	0.973	0.868	35
	4	0.972	0.964	50
	5	0.949	0.678	50
	6	0.988	1.096	50
	7	0.966	1.358	72
	8	0.954	1.298	72
	9	0.945	1.810	100
4	1	0.987	0.614	12.5
	2	0.985	0.883	25
	3	0.961	0.745	25
	4	0.980	0.963	30

Comp i	Element version j	A_{ij}	Cost C_{ij}(mln \$)	Nominal performance $W_{ij}(\%)$
	5	0.958	0.885	30
	6	0.974	1.338	45
	7	0.982	1.445	45

TAB. 2.1: Data of the system elements available in the market [56]

$W_k^0(\%)$	100	80	50	20
$T_k(h)$	4260	800	1200	2496

TAB. 2.2: Parameters of the cumulative load demand curve [56]

Component	m_1	m_2	γ_1	γ_2
1	3	5	0.90	0.80
2	3	3	0.85	0.85
3	3	3	0.95	0.95
4	2	6	0.95	0.90

TAB. 2.3: Parameters of the cost function with discounts [56]

Problem 2

The problem in [50] consists of five components connected in series, and for each component, from 4 to 9 different elements types are available (see Table 2.4 for data). The reliability index A_0 has been set to three different values, namely 0.975, 0.98 and 0.99, to create three variations of the problem. As shown in Table 2.5, the system supply requirements are characterized by a cumulative load demand curve with four different levels. No discount is allowed.

Comp i	Element version j	A_{ij}	Cost C_{ij}(mln \$)	Nominal performance $W_{ij}(\%)$
1	1	0.980	0.590	120
	2	0.977	0.535	100
	3	0.982	0.470	85
	4	0.978	0.420	85
	5	0.983	0.400	48

Comp i	Element version j	A_{ij}	Cost C_{ij}(mln \$)	Nominal performance W_{ij}(%)
	6	0.920	0.180	31
	7	0.984	0.220	26
2	1	0.995	0.205	100
	2	0.996	0.189	92
	3	0.997	0.091	53
	4	0.997	0.056	28
	5	0.998	0.042	21
3	1	0.971	7.525	100
	2	0.973	4.720	60
	3	0.971	3.590	40
	4	0.976	2.420	20
4	1	0.977	0.180	115
	2	0.978	0.160	100
	3	0.978	0.150	91
	4	0.983	0.121	72
	5	0.981	0.102	72
	6	0.971	0.096	72
	7	0.983	0.071	55
	8	0.982	0.049	25
	9	0.977	0.044	25
5	1	0.984	0.986	128
	2	0.983	0.825	100
	3	0.987	0.490	60
	4	0.981	0.475	51

TAB. 2.4: Data of the system elements available in the market [50]

$W_k^0(\%)$	100	80	50	20
$T_k(h)$	4203	788	1228	2536

TAB. 2.5: Parameters of the cumulative load demand curve [50]

Problem 3

The problem in [51] consists of four components connected in series, and for each component, from 4 to 6 different elements types are available. The data of this problem are presented in Table 2.6. The availability index A_0 has been set to three different values, namely 0.9, 0.96 and 0.99, to create three variations of the problem. The system supply requirements are characterized by a cumulative load demand curve with three different levels (Table 2.7). No discount is allowed.

Comp i	Element version j	A_{ij}	Cost C_{ij}(mln \$)	Nominal performance $W_{ij}(\%)$
1	1	0.970	0.520	50
	2	0.964	0.620	80
	3	0.980	0.720	80
	4	0.969	0.890	100
	5	0.960	1.020	150
2	1	0.967	0.516	20
	2	0.914	0.916	50
	3	0.960	0.967	50
	4	0.953	1.367	75
3	1	0.959	0.214	60
	2	0.970	0.384	90
	3	0.959	0.534	180
	4	0.960	0.614	200
	5	0.970	0.783	200
	6	0.960	0.813	240
4	1	0.989	0.683	25
	2	0.979	0.645	25

Comp i	Element version j	A_{ij}	Cost C_{ij}(mln \$)	Nominal performance W_{ij}(%)
	3	0.980	0.697	30
	4	0.960	1.190	70
	5	0.980	1.260	70

TAB. 2.6: Data of the system elements available in the market [51]

W_k^0(%)	100	80	40
$T_k(h)$	20	30	50

TAB. 2.7: Parameters of the cumulative load demand curve [51]

Problem 4

This is a new problem instance which we have randomly generated. The size of this fourth benchmark is larger than the others three existing benchmarks described above. The studied system consists of six components connected in series, and for each component, from 4 to 11 different elements types are available. The availability index A_0 has been set to three different values, namely 0.975, 0.98 and 0.99, to create three variations of the problem. Table 2.8 presents the problem data used for the three examples. No discount is allowed. The data of piecewise cumulative load demand curve are the same as in Table 2.5.

Comp i	Element version j	A_{ij}	Cost C_{ij}(mln \$)	Nominal performance W_{ij}(%)
1	1	0.932	1.590	27.3
	2	0.998	0.515	27.7
	3	0.983	0.225	49.8
	4	0.927	3.220	52.5
	5	0.959	4.020	62.0
	6	0.955	4.270	66.4
	7	0.984	3.670	84.6
	8	0.918	4.630	90.7
	9	0.939	1.010	97.0
	10	0.988	0.779	124
	11	0.984	3.130	129

Comp i	Element version j	A_{ij}	Cost C_{ij}(mln \$)	Nominal performance W_{ij}(%)
2	1	0.989	0.050	35.9
	2	0.923	1.290	44.7
	3	0.900	0.204	51.4
	4	0.946	2.220	63.2
	5	0.917	0.872	68.8
	6	0.962	1.830	81.8
	7	0.994	0.294	82.0
	8	0.984	2.810	115
3	1	0.931	3.620	34.7
	2	0.950	0.475	41.0
	3	0.911	1.170	41.4
	4	0.956	0.793	43.6
	5	0.966	3.740	48.6
	6	0.992	4.590	59.6
	7	0.929	1.740	66.2
	8	0.968	1.720	91.9
	9	0.901	1.300	121
4	1	0.915	2.490	25.1
	2	0.908	0.078	28.8
	3	0.928	1.370	50.2
	4	0.944	4.470	129
5	1	0.908	1.550	34.9
	2	0.980	4.920	64.3
	3	0.964	2.650	108
	4	0.924	4.720	126
6	1	0.965	3.220	24.8
	2	0.927	2.890	47.3
	3	0.986	3.410	58.8
	4	0.983	1.920	107

Comp i	Element version j	A_{ij}	Cost C_{ij}(mln \$)	Nominal performance W_{ij}(%)
	5	0.991	4.510	120
	6	0.954	4.580	125

TAB. 2.8: Randomly generated data for the new problem

Proposed notation for benchmarks

Each benchmark for the RAP of series-parallel MSS will be denoted by $xxxa-(b/c)-d$, where: xxx are the first three characters of the first author's name in the paper where the instance was first introduced; a is the number of components connected in series; (b/c) means that (from b to c) different elements types are available in the market; and d is the number of levels in the cumulative load demand curve. Following this notation, instances of problems 1, 2, 3 and 4 are denoted, respectively, by lis4-(7/11)-4, lev5-(4/9)-4, lev4-(4/6)-3 and ouz6-(4/11)-4.

Size of the search space and number of subspaces

The total number of different solutions to be examined and the number of subspaces are simply given by the following equations:

$$\text{Size of the search space} = \prod_{i=1}^{i=s} \max(J_i)\max(X_i). \tag{2.33}$$

$$\text{Number of subspaces} = \sum_{i=1}^{i=s} \max(J_i) + \max(X_i). \tag{2.34}$$

For example, if the maximum X_i allowed (i.e. the upper bound of X_i) is $\max(X_i) = 10\ \forall i,\ 1 \leq i \leq s$, the size of the search space and the number of subspace are given, for each benchmark, in Table 2.9.

	lev4-(4/6)-3	lis4-(7/11)-4	lev5-(4/9)-4	ouz6-(4/11)-4
Size of the search space	6×10^6	4.85×10^7	5.04×10^8	7.60×10^{10}
Number of subspaces	60	74	79	102

	q	100	5000	5000	400
Parameters values for TS	$mnli$	200	500	450	35
	$mngi$	500	4000	2500	100
	q	10000	10000	10000	10000
	P_m	1	1	1	1
Parameters values for GA	N_s	50	50	50	50
	N_{rep}	1000	2000	2000	2000
	N_c	10	500	500	50

TAB. 2.9: Size of the search space, number of subspaces and parameters values for each instance

2.5.2 Errata for [51, 56]

The results found by GA, as published in [50, 51, 56], are reported in Table 2.10. In these results, there have been some errors in the values of $A(Y)$. For each structure Y obtained by GA in [50, 51, 56], we have recalculated $A(Y)$. In Table 2.10, the incorrect solutions are marked as '*' while the correct values, given in the last column, are infeasible. The errata as given in Table 2.10 were validated in [47] where it was confirmed that "the reason is that the initial data had four digits, however in the tables they were rounded to three digits. Unfortunately, the initial four-digit data are not available at present".

Problem name	A_0	$A(X)$	$C(X,J)$ (mln \$)	J	X	Correct values of $A(X,J)$
	0.900	**0.910***	5.846	4,3,1,4	1,2,3,2	**0.8924**
lev4-(4/6)-3	0.960	**0.961***	6.924	1,3,1,3	3,2,3,4	**0.9416**
	0.990	**0.992***	8.175	1,2,1,2	3,3,3,5	**0.9834**
	0.910	**0.9102***	14.705	11,7,2,2	1,1,4,4	**0.8900**
	0.920	**0.9338***	14.889	11,7,2,3	1,1,4,5	**0.9136**
	0.940	**0.9401***	15.315	11,7,2,3	1,1,5,5	**0.9200**
	0.950	**0.9512***	16.159	10,7,2,2	1,1,5,5	**0.9330**
lis4-(7/11)-4	0.960	**0.9601***	17.900	1,7,2,2	5,1,5,5	**0.9457**
	0.970	**0.9717***	21.578	10,3,2,3	1,3,4,5	**0.9680**
	0.980	**0.9801***	22.470	11,3,2,2	1,3,5,5	**0.9745**

	0.990	**0.9907***	24.304	1,3,5,2	5,3,3,5	**0.9911**
	0.999	**0.9994***	27.398	1,3,2,3	6,4,6,7	**0.9998**
	0.975	0.977	16.450	2,3,2,7,2	2,2,3,3,1	–
lev5-(4/9)-4	0.980	0.981	16.520	2,5,2,7,2	2,6,3,3,1	–
	0.990	0.994	17.050	2,3,2,7,4	2,2,3,3,3	–

TAB. 2.10: Results obtained by genetic algorithm as published in [50, 51, 56] and corrected values of $A(X, J)$

2.5.3 Results obtained by TS

Preliminary numerical tests were used to set the values of tabu search parameters. For each problem instance, others data are randomly generated and used to calibrate the parameters. Once the values of the parameters are set for these preliminary problems data, they are used for the variations of the problems instances to be solved in this paper. In this way, we avoid any parameter (q, $mnli$,$mngi$) overfitting. The parameters values are presented in Table 2.9. Table 2.11 shows the results obtained for the 18 test cases by using TS. The running time did not exceed 127 seconds for any optimization problem.

Problem name	A_0	$A(X,J)$	$C(X,J)$ (mln \$)	J	X	CPU
	0.900	0.9102	5.986	4,3,1,5	1,2,3,2	05.06
lev4-(4/6)-3	0.960	0.9609	7.303	2,3,1,5	2,3,3,2	05.18
	0.990	0.9917	8.328	1,3,1,2	3,3,3,5	05.03
	0.910	0.9136	14.886	11,7,2,3	1,1,4,5	56.59
	0.920	0.9202	15.075	10,7,2,3	1,1,4,5	65.19
	0.940	0.9421	17.805	1,7,5,2	5,1,3,5	56.47
	0.950	0.9518	20.049	1,3,2,2	5,2,5,5	50.18
lis4-(7/11)-4	0.960	0.9604	21.155	10,5,2,3	1,3,4,5	71.15
	0.970	0.9711	21.907	10,3,5,3	1,3,3,5	72.77
	0.980	0.9815	22.656	10,3,2,2	1,3,5,5	78.45
	0.990	0.9911	24.305	1,3,5,2	5,3,3,5	62.85
	0.999	0.9992	26.952	1,3,2,3	6,4,6,6	69.26

	0.975	0.9774	16.450	2,3,2,7,2	2,2,3,3,1	52.91
lev5-(4/9)-4	0.980	0.9808	16.520	2,5,2,7,2	2,6,3,3,1	102.36
	0.990	0.9937	17.050	2,3,2,7,4	2,2,3,3,3	86.57
	0.975	0.9790	11.241	3,1,2,2,3,4	4,4,5,7,2,1	112.71
ouz6-(4/11)-4	0.980	0.9802	11.369	3,1,2,2,3,4	4,5,5,8,2,1	126.49
	0.990	0.9902	12.764	3,1,2,2,3,4	4,4,4,8,2,2	124.76

TAB. 2.11: Results obtained by TS

2.5.4 Comparing the proposed TS with GA

To compare the proposed TS to the GA in references [50, 51, 56], we re-implement this GA in the same conditions (computer, programming language, operating systems, etc.) as for the TS algorithm. On the one hand, this allows us to determine the best feasible GA solutions. On the other hand, it will be possible to solve by GA the fourth problem instance, which is a larger benchmark newly generated in this paper.

Results obtained by GA

Preliminary numerical tests were used to tune the user-specified parameters. These parameters were varied to establish the values most beneficial to the optimization process. Based on these initial experiments the values found to be most appropriate are defined in Table 2.9. These parameters are verified to be identical to those in [50, 51], while the parameters in [56] are not given by the authors. Table 2.12 shows the results obtained for the 18 test cases by using GA. The running time did not exceed 614 seconds for any optimization problem.

Problem name	A_0	$A(X,J)$	$C(X,J)$ $(mln\ \$)$	J	X	CPU
	0.900	0.9115	6.348	1,3,1,1	2,2,3,4	02.75
lev4-(4/6)-3	0.960	0.9651	7.571	2,3,1,3	2,3,3,4	02.76
	0.990	0.9917	8.328	1,3,1,2	3,3,3,5	03.06
	0.910	0.9122	15.812	1,7,2,3	4,1,4,5	67.10
	0.920	0.9258	16.035	5,7,2,3	2,1,4,5	64.22

	0.940	0.9421	17.805	1,7,5,2	5,1,3,5	67.46
	0.950	0.9518	20.049	1,3,2,2	5,2,5,5	70.85
lis4-(7/11)-4	0.960	0.9604	21.155	10,5,2,3	1,3,4,5	84.76
	0.970	0.9711	21.907	10,3,5,3	1,3,3,5	89.05
	0.980	0.9815	22.656	10,3,2,2	1,3,5,5	74.94
	0.990	0.9911	24.305	1,3,5,2	5,3,3,5	83.18
	0.999	0.9992	26.952	1,3,2,3	6,4,6,6	90.24
	0.975	0.9774	16.450	2,3,2,7,2	2,2,3,3,1	427.29
lev5-(4/9)-4	0.980	0.9808	16.520	2,5,2,7,2	2,6,3,3,1	471.34
	0.990	0.9941	17.095	2,3,2,7,3	2,2,3,3,3	493.89
	0.975	0.9790	11.241	3,1,2,2,3,4	4,4,5,7,2,1	598.24
ouz6-(4/11)-4	0.980	0.9844	12.608	3,1,2,2,3,4	4,4,4,6,2,2	613.83
	0.990	0.9902	12.764	3,1,2,2,3,4	4,4,4,8,2,2	439.35

TAB. 2.12: Solutions obtained by GA

Comparisons

The percent that one solution improves upon another is defined in terms of objective function and CPU time as:

$$\text{MPCI} = 100\% \times \frac{\text{(Minimal GA Cost - Minimal TS Cost)}}{\text{Minimal GA Cost}}. \tag{2.35}$$

$$\text{MPTI} = 100\% \times \frac{\text{(Minimal GA Time - Minimal TS Time)}}{\text{Minimal GA Time}}. \tag{2.36}$$

The comparison results are given in Table 2.13 and indicate that the proposed TS generally outperforms the GA. In terms of the solution quality, TS obtained lower cost in 6 of the 18 test cases and is equal to GA in the other 12 cases, as shown in Figure 2.1. In terms of the execution time, TS is quicker than GA in 13 of the 18 test cases and it is less efficient in the other 5 cases. For problems lev5-(4/9)-4, assuming mixing of components is not allowed, our solutions are better in one case and equal in

2 cases. However, reference [50] presented also the solutions when component mixing is permitted. In this paper, assumption 3 stated that mixing of components is not allowed. Therefore, TS results are compared only with the results obtained by [50] without components mixing.

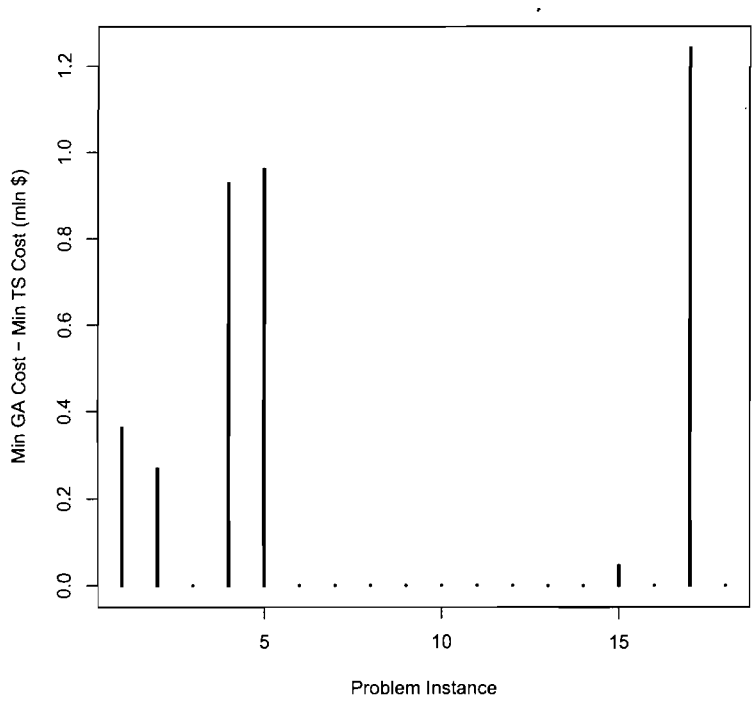


FIG. 2.1 – Comparison of the best TS cost with the best GA cost

		Cost			Running time		
Problem name	A_0	*TS*	*GA*	*MPCI*	*TS*	*GA*	*MPTI*
	0.900	**5.986**	6.348	05.70	05.06	02.75	-84.00
lev4-(4/6)-3	0.960	**7.303**	7.571	03.54	05.18	02.76	-87.68
	0.990	8.328	8.328	00.00	05.03	03.06	-64.38
	0.910	**14.886**	15.812	05.86	56.59	67.10	15.66
	0.920	**15.075**	16.035	05.99	65.19	64.22	-01.51
	0.940	17.805	17.805	00.00	56.47	67.46	16.29
	0.950	20.049	20.049	00.00	50.18	70.85	29.17
lis4-(7/11)-4	0.960	21.155	21.155	00.00	71.15	84.76	16.06

	0.970	21.907	21.907	00.00	72.77	89.05	18.28
	0.980	22.656	22.656	00.00	78.45	74.94	-04.68
	0.990	24.305	24.305	00.00	62.85	83.18	24.44
	0.999	26.952	26.952	00.00	69.26	90.24	23.25
	0.975	16.450	16.450	00.00	52.91	427.29	87.62
lev5-(4/9)-4	0.980	16.520	16.520	00.00	102.36	471.34	78.28
	0.990	**17.050**	17.095	00.26	86.57	493.89	82.47
	0.975	11.241	11.241	00.00	112.71	598.24	81.16
ouz6-(4/11)-4	0.980	**11.369**	12.608	09.83	126.49	613.83	79.39
	0.990	12.764	12.764	00.00	124.76	439.35	71.60

TAB. 2.13: The comparison results

2.5.5 More comparisons of TS and GA

Table 2.14 shows the results obtained for the 18 test cases by using GA and TS for four different running times: 10 seconds, 100 seconds, 1000 seconds and 2000 seconds.

Problem name		10 seconds		100 seconds		1000 seconds		2000 seconds	
	A_0	TS	GA	TS	GA	TS	GA	TS	GA
	0.900	**05.986**	06.348	**05.986**	06.196	**05.986**	06.196	**05.986**	06.196
lev4-(4/6)-3	0.960	**07.303**	07.571	**07.303**	07.571	**07.303**	07.571	**07.303**	07.571
	0.990	**08.328**	**08.328**	**08.328**	**08.328**	**08.328**	**08.328**	**08.328**	**08.328**
	0.910	21.948	15.812	**14.886**	15.812	**14.886**	15.075	**14.886**	15.075
	0.920	23.266	16.035	**15.075**	16.035	**15.075**	**15.075**	**15.075**	**15.075**
	0.940	24.870	19.266	**17.805**	**17.805**	**17.805**	**17.805**	**17.805**	**17.805**
	0.950	24.870	20.879	**20.049**	**20.049**	**20.049**	**20.049**	**20.049**	**20.049**
lis4-(7/11)-4	0.960	24.870	21.242	**21.155**	**21.155**	**21.155**	**21.155**	**21.155**	**21.155**
	0.970	25.876	22.823	**21.907**	**21.907**	**21.907**	**21.907**	**21.907**	**21.907**
	0.980	25.876	23.697	**22.656**	**22.656**	**22.656**	**22.656**	**22.656**	**22.656**
	0.990	26.715	**24.305**	**24.305**	**24.305**	**24.305**	**24.305**	**24.305**	**24.305**
	0.999	29.230	**26.952**	**26.952**	**26.952**	**26.952**	**26.952**	**26.952**	**26.952**
	0.975	16.566	16.544	**16.450**	**16.450**	**16.450**	**16.450**	**16.450**	**16.450**
lev5-(4/9)-4	0.980	16.648	16.541	**16.520**	16.541	**16.520**	**16.520**	**16.520**	**16.520**
	0.990	17.180	17.125	**17.050**	17.095	**17.050**	17.095	**17.050**	17.095
	0.975	24.982	17.410	**11.241**	12.608	**11.241**	**11.241**	**11.241**	**11.241**
ouz6-(4/11)-4	0.980	21.929	16.250	**11.369**	12.996	**11.369**	12.608	**11.369**	12.608
	0.990	36.708	23.748	**12.764**	16.227	**12.764**	**12.764**	**12.764**	**12.764**

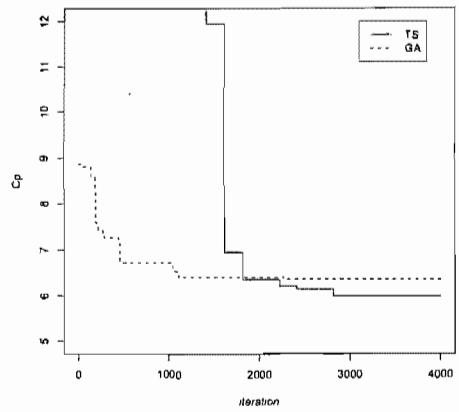
TAB. 2.14: The results obtained by GA and TS for different running times

The convergence curves of TS and GA were drawn for all the 18 test cases. Figures 2.2-2.5 show these curves for problems 1-4. Table 2.14 and the convergence curves show clearly that the proposed TS generally out-performs GA solutions, both in terms of the solution quality and the execution time. From Table 2.14, we remark for example that after 100 seconds, 9 solutions found by TS are better than GA solutions, while the rest of solutions are identical. From Figures 2.2–2.5, we remark that even GA solutions are generally better during the first iterations of the search process, TS solutions become quickly better. Moreover, TS is faster than GA for large problems.

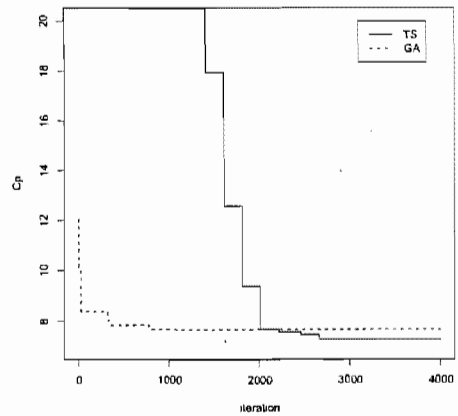
2.6 Conclusion

Tabu search is a powerful algorithmic approach that has been applied with great success to many difficult combinatorial problems. In this paper, we have proposed a tabu search heuristic to solve a relevant problem in reliability design optimization, namely the redundancy allocation problem of homogenous series-parallel multi-state systems. An originality of our approach is that it proceeds by dividing the search space into a set of disjoint subsets, and then by applying TS to each subspace. When compared to GA, TS results in a superior performance in terms of solution quality and CPU time. Furthermore, the random mechanisms of crossover and mutation of GA are avoided by our TS approach, which begins the search, at each run, from the same initial solution, eliminating run to run variability. By re-implementing the GA, we have not only shown the greater efficiency of our TS, but we have re-proven also that the GA proposed in [50, 51, 56] is effective for the RAP of series-parallel MSS. While GA remains at the basis of many effective optimization tools in reliability engineering, TS applied to well-partitioned search spaces seems to be a very promising general approach for other NP-hard reliability design problems. Currently, we are working on the extension of this work to non-homogenous series-parallel multi-state systems.

$A_0 = 0.900$



$A_0 = 0.960$



$A_0 = 0.990$

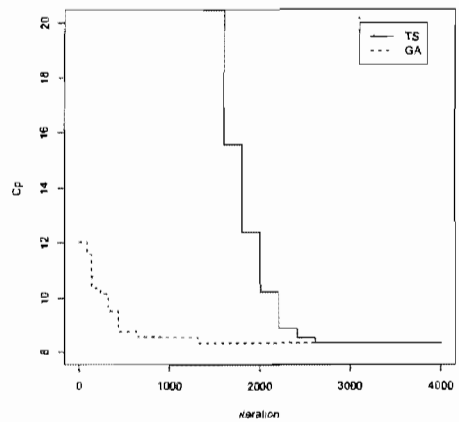
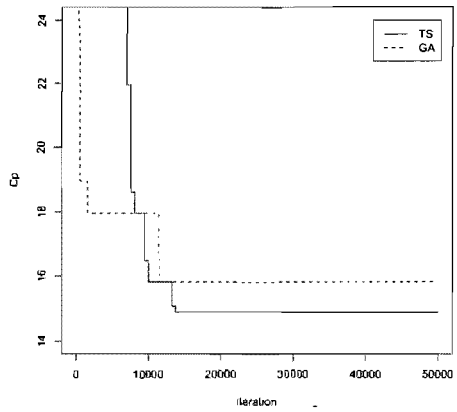
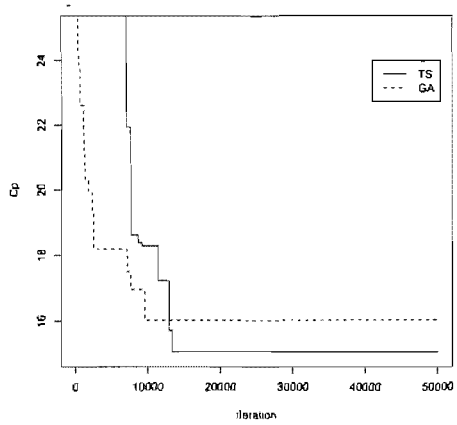


FIG. 2.2 – Convergence curves for GA and TS for lev4-(4/6)-3

$A_0 = 0.910$



$A_0 = 0.920$



$A_0 = 0.940$

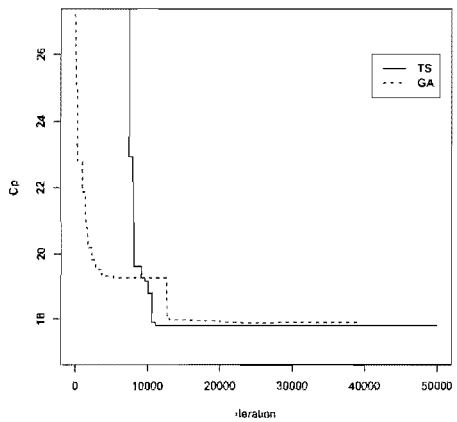
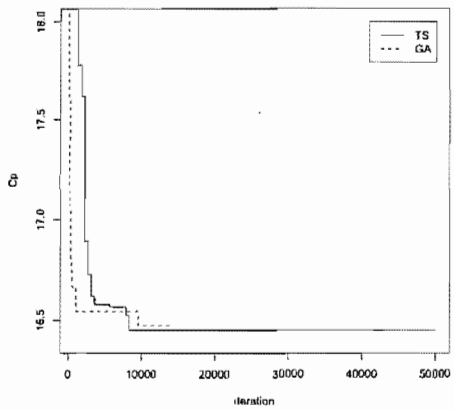
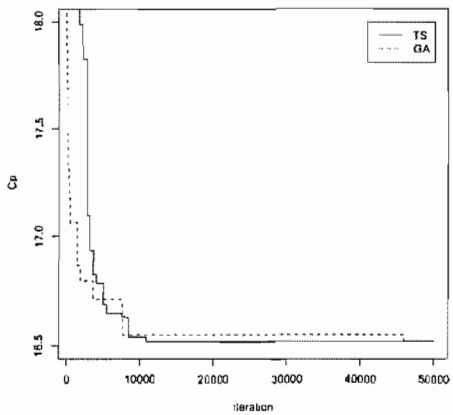


FIG. 2.3 – Convergence curves for GA and TS for lis4-(7/11)-4

$A_0 = 0.975$



$A_0 = 0.980$



$A_0 = 0.990$

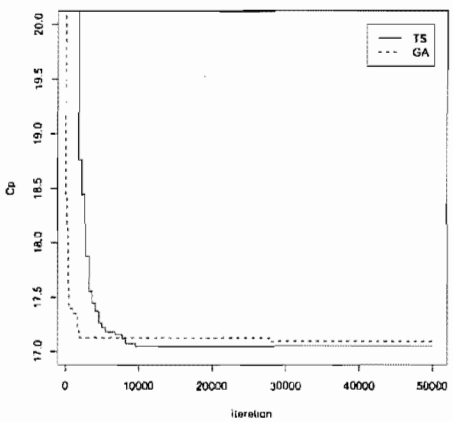
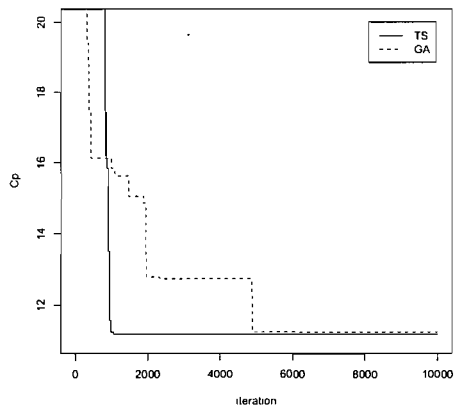
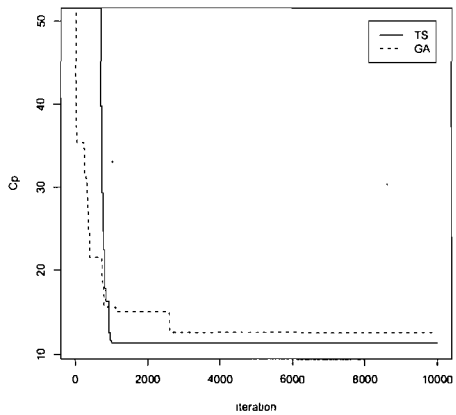


FIG. 2.4 – Convergence curves for GA and TS for lev5-(4/9)-4

$A_0 = 0.975$



$A_0 = 0.980$



$A_0 = 0.990$

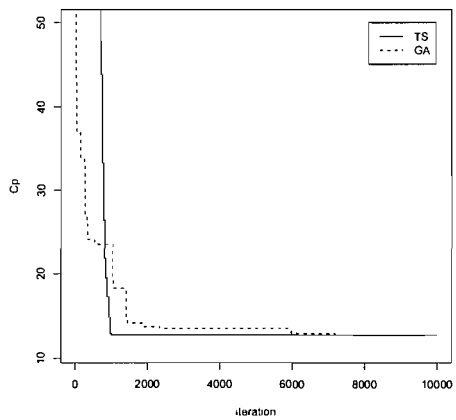


FIG. 2.5 – Convergence curves for GA and TS for ouz6-(4/11)-4

Chapter 3

A heuristic method for non-homogeneous redundancy optimization of series-parallel multi-state systems

Résumé. La principale contribution de cet article réside dans la proposition d'un nouvel algorithme pour résoudre le problème d'allocation de la redondance dans un système multi-états série-parallèle *non homogène*. Des composantes non identiques peuvent être utilisées en parallèle afin d'améliorer la disponibilité du système. Ces composantes sont caractérisées par leur coût, leur performance et leur disponibilité. Notre objectif est de minimiser la somme des coûts d'investissement, tout en satisfaisant la contrainte de disponibilité du système. La disponibilité d'un système multi-états est définie comme la capacité de satisfaire la demande, représentée par une courbe de charge cumulative par morceaux. En autorisant l'utilisation de différentes versions de composantes dans le même sous-système, on peut obtenir une solution qui offre un niveau de disponibilité souhaitable avec un coût inférieur au coût d'une solution utilisant uniquement des composantes identiques en parallèle.

L'approche que nous proposons combine deux métaheuristiques : l'algorithme de recherche tabou et l'algorithme génétique, en plus de l'idée de partitionnement de l'espace global de recherche en plusieurs sous-espaces disjoints. Cette approche est appelée PE/TG (désignant Partitionnement de l'Espace/Tabou-Génétique). Une série d'expérimentations a été effectuée et une étude comparative avec les meilleurs résultats publiés montre l'efficacité de PE/TG, tant au niveau du temps de calcul qu'au niveau de la qualité des solutions obtenues.

A heuristic method for non-homogeneous redundancy optimization of series-parallel multi-state systems

Mohamed OUZINEB
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT),
Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Mustapha NOURELFATH
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT),
Département de génie mécanique,
Université Laval, Local 3344, Pavillon Adrien-Pouliot Sainte-Foy,
Québec G1K 7P4, Canada.

Michel GENDREAU
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT),
Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Article soumis pour publication dans *Journal of Heuristics*

January 2009

Abstract. This paper develops an efficient heuristic to solve the *non-homogeneous* redundancy allocation problem for multi-state series-parallel systems. Non identical components can be used in parallel to improve the system availability by providing redundancy in subsystems. Multiple component choices are available for each subsystem. The components are binary and chosen from a list of products available on the market, and are characterized in terms of their cost, performance and availability. The objective is to determine the minimal-cost series-parallel system structure subject to a multi-state availability constraint. System availability is represented by a multi-state availability function, which extends the binary-state availability. This function is defined as the ability to satisfy consumer demand that is represented as a piecewise cumulative load curve. A fast procedure is used, based on universal generating function, to evaluate the multi-state system availability. The proposed heuristic approach is based on a combination of space partitioning, genetic algorithms (GA) and tabu search (TS). After dividing the search space into a set of disjoint subsets, this approach uses GA to select the subspaces, and applies TS to each selected subspace. The design problem, solved in this study, has been previously analyzed using GA. Numerical results for the test problems from previous research are reported, and larger test problems are randomly generated. These results show that the proposed approach is efficient both in terms of both of solution quality and computational time, as compared to existing approaches.

Keywords. Non-homogeneous redundancy optimization, Multi-state, Series-parallel systems, Meta-heuristics

3.1 Introduction

The redundancy allocation problem (RAP) involves selection of components and levels of redundancy to maximize system performance. The RAP is NP-hard [14]. It has attracted considerable attention from the research community. The great majority of the existing papers on the RAP use traditional binary-state reliability. It is assumed in binary-state reliability modeling that a system and its components may experience only two possible states: good and failed. The RAP for binary-state series-parallel systems has been studied in many different forms, and by considering numerous approaches and techniques. It has been solved by using optimization approaches and techniques such as dynamic programming, integer programming, mixed-integer non-linear programming, heuristics and meta-heuristics: see for example [42, 43, 77] for an extensive overview of these techniques.

However, in many real-life situations, this binary-state representation may not be adequate, since the system and its components may rather have more than two levels of performance varying from perfect functioning to complete failure. A multi-state system (MSS) may perform at different intermediate states between working perfectly and total failure. A series-parallel system consisting of different binary-state components that have a cumulative effect on the entire system performance may be considered as a MSS [55].

The basic concepts of MSS reliability were first introduced in [7, 19, 60, 74]. These works defined the system structure function and its properties. They also introduced the notions of minimal cut set and minimal path set in MSS context, and studied the notions of coherence and component relevancy. A literature review on MSS reliability can be found for example in [55]. The methods currently used for MSS reliability estimation are generally based on four different approaches: (i) the structure function approach which extends Boolean models to the multi-valued case (e.g., [7, 19, 74]); (ii) the Monte-Carlo simulation technique (e.g., [90]); (iii) the Markov process approach (e.g., [72, 86]); and (iv) the universal moment generating function (UMGF) method (e.g., [46, 80]). These approaches are often used by practitioners, for example in the

field of power systems reliability analysis [10, 55]. In practice, different reliability measures can be considered for MSS evaluation and design [5, 57]. The availability of a repairable MSS is defined by the system ability to meet a customer's demand (required performance level). In power systems for example, it is the ability to provide an adequate supply of electrical energy [10].

The RAP for series-parallel MSS is more recent than that of binary-state systems, and it has been much less studied in the literature. It was first introduced in [81], where the universal generating function method was used for the reliability calculation [80]. Following these works, genetic algorithms (GA) were used for the homogeneous RAP of series-parallel MSS in [51, 56], and extended to the non-homogeneous version of this problem in [50]. The other existing solution methods are limited to the homogeneous case, and they include an ant colony optimization in [68], heuristic algorithms in [1, 73], and a tabu search (TS) approach is [71].

Our TS approach in [71] was shown to be efficient when dealing with the RAP of homogenous series-parallel multi-state systems. This approach proceeds by dividing the search space into a set of disjoint subsets, and then by applying TS to each subset. However, when allowing different versions of the components to be allocated in the same subsystem, the number of subsets is too high that the heuristic in [71] is not able to give good quality solution in a reasonable time. The present paper extends this heuristic to solve the *non-homogeneous* case by adding a GA that selects a limited number of spaces. By applying TS only to the subspaces selected by GA (instead of the high number of possible subspaces), the proposed approach, when tested on problems from previous research and on larger problems randomly generated, proved efficient not only at reducing computing time but also at improving overall solution quality. This hybrid approach is called space partitioning/tabu-genetic (SP/TG), SP and TG being acronyms of Space Partitioning and Tabu-Genetic, respectively. It has been successfully applied in [70] to solve the RAP for binary-state systems and the expansion-scheduling problem of homogeneous series-parallel multi-state systems.

The *non-homogeneous* RAP is important for two reasons [50]: (i) by allowing differ-

ent versions of the components to be allocated in the same subsystem, one can obtain a solution that provides the desired availability level with a lower cost than in the solution with identical parallel components; (ii) in practice, the designer often has to include additional components in the existing system. It may be necessary, for example, to modernize a power system according to new demand levels or according to new reliability requirements. Some subsystems can contain components of versions unavailable at present. In this case, some components with the same functionality but with different parameters should compose the subsystems.

The remainder of the paper is organized as follows. In Section 2, we present a description of the RAP for *non-homogeneous* series-parallel multi-state systems. In Section 3, the solution approach is presented and applied to solve our problem. The test problems and the numerical results are presented in Section 4. Finally, some concluding remarks are given in Section 5.

3.2 Problem formulation

3.2.1 General description

The non-homogenous version of the RAP as defined in [50], considers a system consisting of n subsystems connected in series, such that each subsystem i ($i = 1, 2, ..., n$) can contain a number of different components connected in parallel. Each subsystem has m_i versions of the component types available on the market. For each version j ($j = 1, 2, ..., m_i$) belonging to subsystem i, there are x_{ij} components connected in parallel (x_{ij} is a decision variable). Different versions and number of components may be chosen for any given subsystem. Failed components are repaired and the components availabilities are known. Each component i is characterized, according to its version j, by its availability (A_{ij}), capacity (G_{ij}) and cost (C_{ij}). The capacity is measured as a percentage of nominal total system capacity. The MSS availability is defined as its ability to satisfy consumer demand, which is represented as a piecewise cumulative load curve. Indeed, redundancy allows availability improvement, but it increases the total

investment cost. The objective is to design the system so that the total component procurement cost is minimized, subject to a multi-state system availability constraint. The later is based on prescribed demand levels for different operating time periods, and it is taken to be greater than or equal to the required availability level A_0.

3.2.2 Assumptions

1. The characteristics of each component (*i.e.*, its cost, availability and capacity) are known and deterministic.
2. The states of the components are binary (*i.e.*, either good or failed).
3. The component states are mutually *s*-independent, and the system has a finite number of states that are *s*-independent.
4. Mixing of components is allowed in each subsystem (*i.e.*, *non-homogeneous* redundancy can be used).
5. There exists a steady-state distribution of MSS state probabilities.

3.2.3 Notation

n number of series MSS subsystems

m_i number of available components choices for subsystem i

x_{ij} number of components of type j used in subsystem i

$\mathbf{x}_i$ $(x_{i1}, x_{i2}, ..., x_{im_i})$

$\mathbf{X}$ a string of dimension $L = \sum_{i=1}^{n} m_i$ which defines the entire system structure,

$\mathbf{X} = (x_{11}, x_{12}, ..., x_{1m_1}, x_{21}, x_{22}, ..., x_{2m_2}, ..., x_{n1}, x_{n2}, ..., x_{nm_n})$

M_{ij} maximum number of components of version j in parallel belonging to subsystem i

A_{ij} binary-state availability of component of version j belonging to subsystem i

C_{ij} cost of each component of version j in subsystem i

G_{ij} nominal performance level of component of version j in subsystem i

$C(\mathbf{X})$	total cost of series-parallel MSS
$A(\mathbf{X})$	stationary availability index of the overall multi-state series-parallel system
A_0	a specified minimum required level of system availability index
K	number of partitioned intervals
T	MSS operation period
T_k	a partitioned interval in T, $T = \sum_{k=1}^{K} T_k$
$\mathbf{T}$	vector $(T_k)_{1 \le k \le K}$
W_k^0	required MSS performance level for T_k
$\mathbf{W}^0$	vector $(W_k^0)_{1 \le k \le K}$
R_{ms}	MSS reliability
A_{ms}	MSS stationary availability
W_S	total capacity of the system
$y(t)$	MSS state at time t, $y(t) \in \{1, 2, ..., M\}$, 1 is the worst state and M is the best state
m	number or index of MSS state, $m \in \{1, 2, ..., M\}$, 1 is the worst state and M is the best state
W_m	MSS steady-state performance level associated with m
$W(t)$	output performance level of the MSS at time t, $W(t) \in \{W_1, ..., W_M\}$
p_m	$\lim_{t \to \infty}[Pr(W(t) = W_m)]$
$mnli$	maximum number of local iterations without improvement
q	amplification parameter in the penalized objective function
N_s	number of randomly-constructed solutions in the initial population of GA
N_c	number of genetic cycles
N_{rep}	number of reproduction-selection procedures per genetic cycle
CPU	the running time, given in seconds (time of execution)

3.2.4 Mathematical model

We first formulate the cost of subsystem as a linear function and then give the mathematical formulation of the problem. The cost of subsystem i is generally given

by $\sum_{j=1}^{m_i} x_{ij}C_{ij}$. As in [56, 71], if one has to take into account price discounting, the component cost should be considered as a function of the number of components purchased simultaneously. In this case, the cost of subsystem i is a function of x_{ij}:

$$C(\mathbf{x}_i) = \sum_{j=1}^{m_i} x_{ij}C_{ij}(x_{ij}). \tag{3.1}$$

The total cost can be calculated by the sum of the costs of the chosen components. The *non-homogeneous* redundancy allocation problem studied in this paper can then be stated as follows:

$$\text{minimize } C(\mathbf{X}) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} x_{ij}C_{ij}(x_{ij}) \tag{3.2}$$

subject to

$$A(\mathbf{X}) \geq A_0, \tag{3.3}$$

$$x_{ij} \in \{0, 1, ..., M_{ij}\}, \quad i = 1, 2, ..., n; j = 1, 2, ..., m_i. \tag{3.4}$$

Constraint (3.3) represents the availability constraint. Constraint (3.4) specifies that, for each version j belonging to subsystem i, the number of components connected in parallel is an integer which cannot be higher than a pre-selected maximum number M_{ij}.

3.2.5 Availability of MSS

The series-parallel multi-state system is considered to have a range of performance levels from perfect functioning to complete failure. In fact, the system failure can lead to decreased capability to accomplish a given task, but not to complete failure. For example, in electric power systems, reliability is considered as a measure of the ability of the system to meet the load demand (W^0). The reliability index was defined in [60, 75]. As in [10], using an analogy with the Loss of Load Probability index (LOLP) we can write LOLP $= 1-A(X)$. LOLP is understood as the probability that the system cannot supply a given demand load. It is assumed that there exists a stationary distribution of failures (steady-state). This hypothesis is acceptable and provided by excluding

some phenomena that could affect the distribution of elements, or to consider a long production periods in order to be sure that the system can be in a steady-state. For repairable MSS, a multi-state stationary availability A is used as $Pr(W(t) \geq W^0)$ after enough time has passed for this probability to become constant (the reader is referred for example to [51, 56]). In the steady-state the distribution of states probabilities is given by equation (3.5), while the multi-state stationary availability according to the demand W^0 is formulated by equation (3.6):

$$p_m = \lim_{t \to \infty} [Pr(W(t) = W_m)], \tag{3.5}$$

$$A(\mathbf{X}, W^0) = \sum_{W_m \geq W^0} p_m. \tag{3.6}$$

Since the demand is represented as a piecewise cumulative load curve, the operation period T is divided into K intervals (number of pieces). Each interval has a length T_k and a required demand level W_k^0 $(k = 1, ..., K)$. In this case, the MSS availability index is [51, 56]:

$$A(\mathbf{X}, W^0) = \frac{1}{\sum_{k=1}^{K} T_k} \sum_{k=1}^{K} A(X, W_k^0) T_k. \tag{3.7}$$

The equation (3.7) can be written as follows:

$$A(\mathbf{X}, W^0) = \frac{1}{\sum_{k=1}^{K} T_k} \sum_{k=1}^{K} Pr(W_S \geq W_k^0) T_k, \tag{3.8}$$

where $Pr(W_S \geq W_k^0)$ is the probability that the total capacity of the system (W_S) is not lower than a specific demand level W_k^0.

For solving the combinatorial optimization problem formulated in Section 2.4, it is important to have an effective and fast procedure to evaluate the availability of a series parallel MSS. For this evaluation, we use the universal moment generating function (UMGF), which has been proven to be very effective for high-dimensional combinatorial optimization problems. The reader is referred for example to [55] or [71] for details about the availability evaluation method by the UMGF.

3.3 Solution methodology

The search space S is composed of all possible series-parallel structures. This space is first divided into a set of disjoint subspaces. Then, an efficient TS is applied to each subspace selected by using GA.

3.3.1 Partition of the search space

Each subspace (also called region) is characterized by its own address. Given a system structure as a string of integers $\mathbf{X}$, ($\mathbf{X} = (x_{11}, ..., x_{1m_1}, ..., x_{n1}, ..., x_{nm_n})$), the address of $\mathbf{X}$ is defined by the number of all available components. That is, address($\mathbf{X}$) $=\sum_{i=1}^{n}\sum_{j=1}^{m_i} x_{ij}$. A search subspace of address r, denoted by S_r is defined as the set of series-parallel structures (solutions) which have the same address, equal to r. While the lower bound of r is n, its upper bound is given by $N = \sum_{i=1}^{n}\sum_{j=1}^{m_i} M_{ij}$. Remark that $(S_r)_{2s \leq r \leq N}$ is a partition of the search space S. A quick analysis of the problem clearly indicates that values of r that are very large (close to N) or quite small (close to $2s$) can be expected to yield unattractive solutions. The algorithm could have been designed to take advantage of this. We did not do so, since computational experiments shooed that very little is devoted to the exploration of subspaces with r close to $2s$ or N.

3.3.2 Tabu search

Tabu search is a meta-heuristic that consists in an iterative method, where at each iteration we move from a current solution to a new solution in a neighbourhood, until some stopping criterion has been satisfied. Our neighbourhood structure is defined as follows: at each iteration of TS, the local transformations (or moves), that can be applied to the current solution $\mathbf{X}$, define a set of neighbouring solutions for a given subspace as: Neighborhood($\mathbf{X}$) = series-parallel structures obtained by applying a single move to $\mathbf{X}$. The move applied to $\mathbf{X}$ consists in changing the number of components in parallel by adding and subtracting one, if possible, for any subsystem The move applied to $\mathbf{X}$

consists in changing the number of components in parallel by adding and subtracting one, if possible, for any subsystem ($x_{i_1j_1} \rightarrow x_{i_1j_1} + 1$ and $x_{i_2j_2} \rightarrow x_{i_2j_2} - 1$). It follows that address $\mathbf{X}$ does not change after a local transformation of $\mathbf{X}$ and *the search process remains in the same subspace.* TS enhances the local search performance by using memory structures: once a potential solution has been determined, it is marked as *tabu*, so that the algorithm does not visit that possibility repeatedly. That is, tabus are used to prevent from cycling when moving away from local optima through non-improving moves. At each iteration, the best solution $\mathbf{X}$' in a subset $V(\mathbf{X})$ Neighborhood($\mathbf{X}$) is selected and considered as a tabu solution for some next iterations. The subset $V(\mathbf{X})$, called the effective neighborhood), is generated by eliminating the tabu solutions from Neighborhood($\mathbf{X}$). Tabu solutions are stored in a short-term memory, called tabu list, which contains the solutions that have been visited in the recent past. The size of the tabu list (tabu tenure) used in this paper is dynamic, as it is usually found that using a variable size tabu list is more efficient [22, 71]. Furthermore, the objective function (to be minimized) is penalized by adding to $C(\mathbf{X})$ the term $\alpha \max(0, A_0 - A(\mathbf{X}))$, where α is a sufficiently large number. This term represents a penalty for constraint violation, which forces the TS algorithm to consider only feasible solutions. Finally, the stopping criterion is specified in terms of a maximum number of local iterations (mnli) without improving the best-known solution.

3.3.3 Hybrid optimization method

The proposed SP/TG approach is based on a combination of space partitioning, genetic algorithms and tabu search. After dividing the search space into a set of disjoint subsets, this approach uses GA to select the subspaces, and applies TS to each selected subspace. The GA used in this paper is based on the method commonly known as the steady-state GA or GENITOR [84]. Starting from a randomly-generated initial population, the genetic algorithm applies iteratively four operators, selection, crossover, mutation and culling to produce a new population of solutions. Compared to standard genetic algorithm, in SP/TG, the mutation step is replaced by Tabu Search. These operators can be summarized as follws:

1. **Selection operator:** it selects randomly from the populations two solutions named parent 1 and parent 2 that will "reproduce" to create a new solution.

2. **Crossover operator:** it produces a new solution (child or offspring) from a selected pair of parent solutions. We use the so-called two-point (or fragment) crossover operator, which, as illustrated in Table 3.1, creates the child string O for the given pair of parent strings X_1 and X_2 by:

   (a) Choosing randomly two positions k and m in the string O;

   (b) Copying string elements belonging to the fragment between k and m from X_1;

   (c) Copying the rest of string elements from X_2.

Parent string X_1	0	0	0	1	3	0	1	0	1	1	0	3
Parent string X_2	**2**	**0**	**2**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**4**	**0**
Child string O	**2**	**0**	0	1	3	0	1	0	**1**	**0**	**4**	**0**

TAB. 3.1: An example illustrating the 2-point crossover procedure

3. **Tabu Search** (this replaces the Mutation operator): it looks for the best solution in the subspace associated with the new child using the TS algorithm described previously. The best solution found becomes the new child.

4. **Culling operator**: : it generates the new population. Each new solution obtained by TS is decoded to obtain its fitness value. These fitness values, which are a measure of quality, are used to compare different solutions. This fitness is evaluated according to the penalized objective function $C(\mathbf{X}) + \alpha \max(0, A_0 - A(\mathbf{X}))$. If the new solution obtained by TS is better than the worst solution currently in the population, it replaces it, else it is discarded. If the population contains equivalent subspaces following a replacement, redundancies are eliminated and the population size decreases consequently. After N_{rep} new solutions have been produced (or if the population contains only one subspace), new randomly con-

structed solutions are generated to replenish the population, and a new genetic cycle begins. The GA is terminated after N_c genetic cycles. The final GA population contains the best selected subspace represented by its best solution found by TS

To avoid the application of TS to any given subspace more than once, a set E (initially empty) is used to store the addresses of subspaces already explored; the TS step is not performed if a child's address belongs to E.

3.4 Computational results

All the algorithms were implemented in C^{++}. The numerical tests were completed on an Intel Pentium IV 3000 MHz DEC station 5000/240 with 1024 Mbytes of RAM running under Linux. All computations use real float point precision without truncating or rounding values.

3.4.1 Test problems

Four design optimization problems (benchmarks) are used to analyze the SP/TG methodology for the non-homogeneous RAP for series-parallel MSS. The first three problems were introduced in [50, 51, 56], while the fourth is a larger instance introduced in [71]. For the tables of input data, the reader is referred to [71] where these four benchmarks were denoted by lis4-(7/11)-4, lev5-(4/9)-4, lev4-(4/6)-3 and ouz6-(4/11)-4. In the notation $xxxa$-(b/c)-d, xxx are the first three characters of the first author's name in the paper where the instance was first introduced; a is the number of subsystems connected in series; (b/c) means that (from b to c) different components types are available in the market; and d is the number of levels in the cumulative load-demand curve. When considering that mixing of components is allowed, the total number of different solutions to be examined and the number of subspaces are given by the following equations:

$$\text{Size of the search space} = \prod_{i=1}^{n} \prod_{j=1}^{m_i} M_{ij} \tag{3.9}$$

$$\text{Number of subspaces} = \sum_{i=1}^{n} \sum_{j=1}^{m_i} M_{ij} - n + 1 \tag{3.10}$$

Let us consider for example that the maximum number of components allowed to reside in parallel is 10 for each subsystem. In this case, Table 3.2 presents, for each benchmark, the size of the search space and the number of subspaces for the homogeneous and non-homogeneous cases. From this table, we remark that the sizes of the problems increase drastically when considering non-homogeneous redundancy. As a result, solution methodologies for homogeneous redundancy, including our heuristic developed in [71], are no longer valid.

	Cases	lev4-(4/6)-3	lev5-(4/9)-4	lis4-(7/11)-4	ouz6-(4/11)-4
Size of the search space	homogeneous	10^6	10^8	10^7	10^{10}
	non-homogeneous	10^{20}	10^{26}	10^{34}	10^{42}
Nbr. of subspaces	homogeneous	60	79	74	102
	non-homogeneous	197	286	337	415

TAB. 3.2: Search space size and disjoint subspaces (maximum of 10 components in parallel)

3.4.2 Parameter settings

Meta-heuristics generally depend on their parameters. Setting these parameters is an important task that can be time-consuming. Regarding this, we would like to underline first that even if the proposed heuristic combines two meta-heuristics, it has the advantage of using only a few parameters that correspond to the termination criterion. Moreover, we have performed our parameters in such a way that we have not to update their values whenever we change the problem data. In fact, preliminary numerical tests were used to set the values of the parameters. For each problem instance, other data are randomly generated and used to calibrate the parameters. Once the values of the parameters are set for these preliminary problems data, they are used for the variations of the problem instances to be solved in this paper. In this way, we avoid any parameter's over-fitting. Parameters' values are presented in Table 3.3. The value of α is set to 10000 for all problems.

	lev4-(4/6)-3	lev5-(4/9)-4	lis4-(7/11)-4	ouz6-(4/11)-4
N_{rep}	100	200	300	200
N_s	500	500	500	500
$mnli$	20	10	20	10
N_c	10	50	50	50

TAB. 3.3: SP/TG parameters values

3.4.3 GA re-implementation for comparison

To the best of our knowledge, the only existing method to solve the *non-homogeneous* RAP for MSS is the GA proposed in [50]. To compare the proposed methodology with this GA, we have re-implemented it in the same conditions (computer, programming language, operating systems, etc.) as for the SP/TG heuristic. On the one hand, this allows us to determine the best feasible GA solutions for the problems that are solved for the homogeneous case in [51, 56]. On the other hand, it will be possible to solve by GA the problem instance in [71], which is a larger benchmark also never solved in the literature for the non-homogeneous case. The values of the GA parameters are presented in Table 3.4. In this table, the notation P_m refers to the mutation probability used in [50]. Note that for the non-homogeneous problem solved in [50], all the parameters are the same as in this reference. Furthermore, the parameters used for the problems newly solved by GA are calibrated using preliminary numerical tests. These parameters were varied to establish the values most beneficial to the optimization process.

	lev4-(4/6)-3	lev5-(4/9)-4	lis4-(7/11)-4	ouz6-(4/11)-4
P_m	1	1	1	1
N_s	500	500	500	500
N_{rep}	1000	2000	2000	2000
N_c	100	200	200	500

TAB. 3.4: GA parameters values

3.4.4 Comparisons of the proposed SP/TG with GA

The percent that one solution improves upon another is defined in terms of objective function and CPU time as:

$$\text{MPCI} = 100\% \times \frac{\text{(Minimal GA Cost - Minimal SP/TG Cost)}}{\text{Minimal GA Cost}}. \quad (3.11)$$

$$\text{MPTI} = 100\% \times \frac{\text{(Minimal GA Time - Minimal SP/TG Time)}}{\text{Minimal GA Time}}. \quad (3.12)$$

Comparing the best solutions and running times over ten runs

Using different random seeds, ten runs were made for each problem instance. The results of the best solutions obtained over ten runs by SP/TG and GA, for each instance, are presented in Tables 3.5 and 3.6. In these tables, each sub-system structure i is represented by an ordered sequence of strings as follows: $1(x_{i1}), 2(x_{i2}), ..., m_i(x_{im_i})$, that are directly taken from the corresponding solution $\mathbf{X} = (x_{11}, x_{12}, ..., x_{1m_1}, x_{21}, x_{22}, ..., x_{2m_2}, ..., x_{n1}, x_{n2}, ..., x_{nm_n})$.

In the comparison results given in Table 3.7, the best solutions and the best running times are independently selected among the ten runs. These results indicate that the proposed SP/TG generally outperforms the GA. In fact, in terms of the solution quality, Table 3.7 shows that SP/TG obtained lower cost in 4 of the 18 test cases, one higher cost, and it is equal to GA in the other 13 cases. In terms of the execution time, SP/TG is quicker than GA for all instances.

Problem name	A_0	$A(\mathbf{X})$	$C(\mathbf{X})$	Best solution					
				1	2	3	4	5	6
	0.900	0.901	5.423	4(1)	3(2)	1(3)	3(1), 5(1)	–	–
lev4-(4/6)-3	0.960	0.963	7.009	1(3)	2(1), 3(2)	1(3)	3(1), 5(1)	–	–
	0.990	0.991	8.180	1(3)	3(3)	1(3)	3(1), 4(2)	–	–
	0.975	0.976	12.855	4(2), 6(1)	5(6)	1(1), 4(1)	7(3)	4(3)	–
lev5-(4/9)-4	0.980	0.981	14.770	4(2), 6(1)	3(2)	2(1), 3(2)	7(3)	3(2), 4(1)	–
	0.990	0.992	15.870	4(2), 6(1)	3(2)	2(2), 3(1)	7(3)	4(3)	–

Problem name	A_0	$A(\mathbf{X})$	$C(\mathbf{X})$	Best solution					
				1	2	3	4	5	6
	0.910	0.914	14.886	11(1)	7(1)	2(4)	3(5)	–	–
	0.920	0.920	15.075	10(1)	7(1)	2(4)	3(5)	–	–
	0.940	0.941	17.418	1(5)	7(1)	2(5)	2(1), 3(4)	–	–
	0.950	0.950	19.861	1(5)	3(2)	2(5)	2(3), 3(2)	–	–
lis4-(7/11)-4	0.960	0.961	20.570	10(1)	2(1), 5(2)	2(4)	2(1), 3(4)	–	–
	0.970	*0.970*	21.288	*10*(1)	2(1), 5(2)	2(5)	2(3), 3(2)	–	–
	0.980	0.981	22.738	1(5)	2(1), 5(2)	2(5)	2(1), 3(4)	–	–
	0.990	0.990	23.779	1(5)	1(1), 5(2)	2(5)	2(5)	–	–
	0.999	0.999	26.919	1(6)	3(4)	2(5)	2(3), 3(3)	–	–
	0.975	0.979	11.241	3(4)	1(4)	2(5)	2(7)	3(2)	4(1)
ouz6-(4/11)-4	0.980	0.980	11.369	3(4)	1(5)	2(5)	2(8)	3(2)	4(1)
	0.990	0.992	12.764	3(4)	1(4)	2(4)	2(8)	3(2)	4(2)

TAB. 3.5: SP/TG best solutions over ten runs

Problem name	A_0	$A(\mathbf{X})$	$C(\mathbf{X})$	Best solution					
				1	2	3	4	5	6
	0.900	*0.900*	*5.803*	*2(1), 3(1)*	*3(2)*	*1(3)*	*3(1), 4(1)*	–	–
lev4-(4/6)-3	0.960	0.960	7.365	2(2)	2(2), 3(1)	1(3)	2(2), 3(1)	–	–
	0.990	0.991	8.180	1(3)	3(3)	1(3)	3(1), 4(2)	–	–
	0.975	0.976	12.855	4(2), 6(1)	5(6)	1(1), 4(1)	7(3)	4(3)	–
lev5-(4/9)-4	0.980	0.981	14.770	4(2), 6(1)	3(2)	2(1), 3(2)	7(3)	3(2), 4(1)	–
	0.990	0.992	15.870	4(2), 6(1)	3(2)	2(2), 3(1)	7(3)	4(3)	–
	0.910	0.914	14.886	11(1)	7(1)	2(4)	3(5)	–	–
	0.920	0.920	15.075	10(1)	7(1)	2(4)	3(5)	–	–
	0.940	0.941	17.418	1(5)	7(1)	2(5)	2(1), 3(4)	–	–
	0.950	0.950	19.861	1(5)	3(2)	2(5)	2(3), 3(2)	–	–
lis4-(7/11)-4	0.960	0.961	20.570	10(1)	2(1), 5(2)	2(4)	2(1), 3(4)	–	–
	0.970	0.970	21.288	10(1)	2(1), 5(2)	2(5)	2(3), 3(2)	–	–
	0.980	0.981	22.562	10(1)	3(3)	2(5)	2(4), 3(1)	–	–
	0.990	0.990	23.779	1(5)	1(1), 5(2)	2(5)	2(5)	–	–
	0.999	0.999	26.919	1(6)	3(4)	2(5)	2(3), 3(3)	–	–
	0.975	0.979	11.241	3(4)	1(4)	2(5)	2(7)	3(2)	4(1)
ouz6-(4/11)-4	0.980	0.980	11.516	3(5)	1(5)	2(5)	2(7)	3(2)	4(1)
	0.990	0.990	12.911	3(5)	1(4)	2(4)	2(7)	3(2)	4(2)

TAB. 3.6: GA best solutions over ten runs

Problem name	A_0	Cost			Running time		
		SP/TG	*GA*	*MPCI*	*SP/TG*	*GA*	*MPTI*
lev4-(4/6)-3	0.900	**5.423**	5.803	06.55	**03.68**	10.03	63.31
	0.960	**7.009**	7.365	04.83	**04.12**	10.29	59.96
	0.990	**8.180**	**8.180**	00.00	**03.68**	10.34	64.41
lev5-(4/9)-4	0.975	**12.855**	**12.855**	00.00	**18.48**	71.01	73.98
	0.980	**14.770**	**14.770**	00.00	**17.32**	61.05	71.63
	0.990	**15.870**	**15.870**	00.00	**21.36**	67.09	68.16
lis4-(7/11)-4	0.910	**14.886**	**14.886**	00.00	**61.46**	166.75	63.14
	0.920	**15.075**	**15.075**	00.00	**56.05**	172.72	67.55
	0.940	**17.419**	**17.418**	00.00	**52.70**	139.48	62.22
	0.950	**19.861**	**19.861**	00.00	**25.37**	137.28	81.52
	0.960	**20.570**	**20.570**	00.00	**46.33**	154.23	69.96
	0.970	**21.288**	**21.288**	00.00	**38.83**	156.68	75.22
	0.980	22.738	**22.562**	-00.77	**62.14**	112.79	44.91
	0.990	**23.779**	**23.779**	00.00	**64.69**	119.73	45.97
	0.999	**26.919**	**26.919**	00.00	**70.20**	197.44	64.44
ouz6-(4/11)-4	0.975	**11.241**	**11.241**	00.00	**77.00**	372.76	79.34
	0.980	**11.369**	11.516	01.28	**66.60**	415.45	83.97
	0.990	**12.764**	12.911	01.14	**45.58**	377.57	87.93

TAB. 3.7: The comparison results (the best costs and running times among ten runs are in bold

Comparing the mean values over ten runs

The mean values obtained by each method (*i.e.*, mean costs and mean CPU over ten runs) are given in Table 3.8. In terms of the mean objective function, SP/TG obtained lower costs in 13 of the 18 test cases, obtained higher costs in 2 of the 18 test cases, and is equal to GA in the other 3 cases. In terms of the mean running time, SP/TG is quicker than GA for all instances.

Problem name	A_0	Cost			Running time		
		SP/TG	*GA*	*MPCI*	*SP/TG*	*GA*	*MPTI*
	0.900	**5.423**	5.803	06.55	**04.59**	10.17	54.87

Problem name	A_0	Cost			Running time		
		SP/TG	*GA*	*MPCI*	*SP/TG*	*GA*	*MPTI*
lev4-(4/6)-3	0.960	**7.009**	7.369	04.88	**04.72**	10.36	54.44
	0.990	**8.193**	8.195	00.02	**04.47**	10.45	57.22
lev5-(4/9)-4	0.975	**12.855**	**12.855**	00.00	**28.68**	74.14	61.32
	0.980	**14.774**	14.778	00.01	**25.51**	69.44	63.26
	0.990	**15.870**	**15.870**	00.00	**25.66**	74.36	65.49
lis4-(7/11)-4	0.910	**14.886**	14.937	00.34	**125.16**	183.17	31.67
	0.920	**15.075**	**15.075**	00.00	**81.51**	210.64	61.30
	0.940	**17.477**	17.500	00.13	**142.27**	174.00	18.24
	0.950	**19.861**	20.363	02.46	**73.87**	180.73	59.13
	0.960	**20.570**	20.802	01.12	**108.99**	174.80	37.65
	0.970	**21.324**	21.424	00.47	**72.71**	194.24	62.57
	0.980	**22.738**	22.781	00.19	**106.31**	194.26	45.27
	0.990	**23.825**	23.867	00.18	**147.81**	151.08	2.16
	0.999	27.391	**26.923**	-01.71	**122.62**	209.41	41.45
ouz6-(4/11)-4	0.975	11.296	**11.241**	-00.49	**101.54**	382.71	73.47
	0.980	**11.401**	11.516	00.10	**96.22**	419.77	77.08
	0.990	**12.779**	12.911	01.02	**96.49**	388.37	75.16

TAB. 3.8: The comparison results in terms of the mean values (ten trials)

Comparing the worst results over ten runs

The worst results obtained by each method (*i.e.*, worst costs and worst CPU among the ten runs) are given in Table 3.9. In terms of the worst objective function, SP/TG obtained lower costs in 8 of the 18 test cases, obtained higher costs in 6 of the 18 test cases, and is equal to GA in the other 4 cases. In terms of the worst running time, SP/TG is quicker than GA in 14 of the 18 test cases and it is less efficient in the other 4 cases.

Problem name	A_0	Cost			Running time		
		SP/TG	*GA*	*MPCI.*	*SP/TG*	*GA*	*MPTI*
	0.900	**5.423**	5.803	06.55	**05.82**	10.34	43.71

Problem name	A_0	Cost			Running time		
		SP/TG	*GA*	*MPCI*	*SP/TG*	*GA*	*MPTI*
lev4-(4/6)-3	0.960	**7.009**	7.403	05.32	**05.84**	10.53	44.54
	0.990	**8.315**	8.328	00.16	**05.66**	10.58	46.56
lev5-(4/9)-4	0.975	**12.855**	**12.855**	00.00	**38.68**	77.64	50.18
	0.980	14.789	**14.780**	-00.06	**33.68**	71.24	52.72
	0.990	**15.870**	**15.870**	00.00	**31.46**	77.96	59.56
lis4-(7/11)-4	0.910	**14.886**	15.218	02.18	406.95	**194.66**	-52.17
	0.920	**15.075**	**15.075**	00.00	**140.52**	225.04	37.56
	0.940	18.004	**17.711**	-01.63	375.92	**199.73**	-46.87
	0.950	**19.861**	20.748	04.27	**174.07**	226.18	23.04
	0.960	**20.570**	21.148	02.73	234.27	**195.37**	-16.60
	0.970	21.556	**21.541**	-00.01	**134.30**	218.10	38.42
	0.980	**22.738**	22.965	00.99	**174.81**	260.10	32.79
	0.990	**24.136**	24.227	00.38	424.76	**172.94**	-59.28
	0.999	29.249	**26.952**	-07.58	**211.19**	228.90	07.74
ouz6-(4/11)-4	0.975	11.794	**11.241**	-04.69	**143.74**	389.33	63.08
	0.980	11.687	**11.516**	-01.46	**142.78**	432.91	67.02
	0.990	**12.911**	**12.911**	00.00	**158.64**	394.81	59.82

TAB. 3.9: The comparison results in terms of the worst values (among ten trials)

Comparing the average results of SP/TG with the best results of GA

From Table 3.10, we conclude that if the average solutions of SP/TG are compared to the best solutions of GA, in 4 instances SP/TG is better, in 8 instances GA is better, and in the remaining instances (6) they are equal. Moreover, when comparing the average running time of SP/TG with the best CPU of GA, in 16 instances SP/TG is better, and in 2 instances GA is better. This comparison of average performances of SP/TG versus best-of-ten GA performances (GA) highlights clearly the efficiency of SP/TG.

Problem name	A_0	Cost			Running time		
		SP/TG	*GA*	*MPCI*	*SP/TG*	*GA*	*MPTI*
lev4-(4/6)-3	0.900	**5.423**	5.803	06.55	**04.59**	10.03	54.24
	0.960	**7.009**	7.365	04.83	**04.72**	10.29	54.13
	0.990	8.193	**8.180**	-00.16	**04.47**	10.34	56.77
lev5-(4/9)-4	0.975	**12.855**	**12.855**	00.00	**28.68**	71.01	59.61
	0.980	14.774	**14.770**	-00.01	**25.51**	61.05	58.21
	0.990	**15.870**	**15.870**	00.00	**25.66**	67.09	61.75
lis4-(7/11)-4	0.910	**14.886**	**14.886**	00.00	**125.16**	166.75	24.94
	0.920	**15.075**	**15.075**	00.00	**81.51**	172.72	52.80
	0.940	17.477	**17.418**	-00.34	142.27	**139.48**	-01.96
	0.950	**19.861**	**19.861**	00.00	**73.87**	137.28	46.19
	0.960	**20.570**	**20.570**	00.00	**108.99**	154.23	29.33
	0.970	21.324	**21.288**	-00.17	**72.71**	156.68	53.59
	0.980	22.738	**22.562**	-00.77	**106.31**	112.79	05.74
	0.990	23.825	**23.779**	-00.19	147.81	**119.73**	-19.00
	0.999	27.391	**26.919**	-01.72	**122.62**	197.44	37.89
ouz6-(4/11)-4	0.975	11.296	**11.241**	-00.49	**101.54**	372.76	72.76
	0.980	**11.401**	11.516	01.00	**96.22**	415.45	76.84
	0.990	**12.779**	12.911	01.02	**96.49**	377.57	74.44

TAB. 3.10: Comparison of mean SP/TG with best GA performances over 10 seeds

To measure the robustness of the SP/TG and GA algorithms, the standard deviations over ten runs are given in Table 3.11 for each instance. We remark from this table that for each instance, the standard deviation is very low for both algorithms. This implies that the proposed method and GA are both robust. Nevertheless, our SP/TG has a lower standard deviation in 13 of the 18 test cases. Therefore, SP/TG can be considered as more robust than GA, while yielding generally solutions with lower costs in a shorter time.

Problem name	A_0	SP/TG	GA
	0.900	00.00	00.00
lev4-(4/6)-3	0.960	00.00	00.02

Problem name	A_0	SP/TG	GA
	0.990	00.04	00.05
	0.975	00.00	00.00
lev5-(4/9)-4	0.980	00.01	00.00
	0.990	00.00	00.00
	0.910	00.00	00.11
	0.920	00.00	00.00
	0.940	00.18	00.11
	0.950	00.00	00.34
lis4-(7/11)-4	0.960	00.00	00.15
	0.970	00.09	00.13
	0.980	00.00	00.20
	0.990	00.11	00.19
	0.999	00.97	00.01
	0.975	00.17	00.00
ouz6-(4/11)-4	0.980	00.10	00.00
	0.990	00.05	00.00

TAB. 3.11: Standard deviations of SP/TG and GA

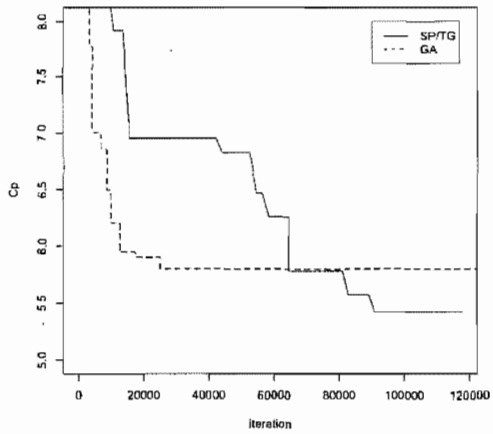
3.4.5 Convergence of SP/TG and GA

The convergence curves of SP/TG and GA were drawn for all the 18 test cases, showing similar behaviour in all these cases. Figures 3.1-3.4 present examples of these curves and show that the proposed SP/TG algorithm requires generally less numbers of iterations than GA. We remark that even if the solutions of our algorithm are generally not as good as GA solutions during the first iterations of the search process, they tend to become quickly better.

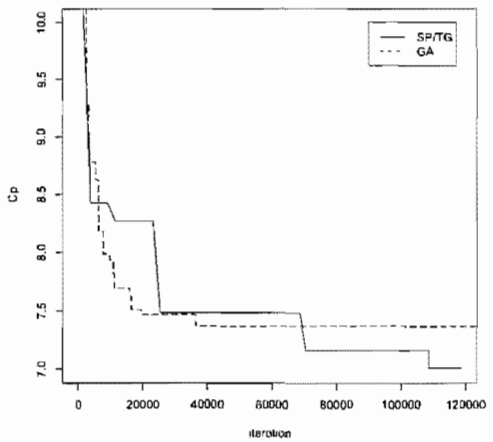
3.5 Conclusion

The non-homogeneous redundancy allocation problem for multi-state series-parallel systems has been efficiently solved in this paper by combining two meta-heuristics, and the idea of space portioning that proceeds by dividing the search space into a set of disjoint regions. A steady-state genetic algorithm has been used for the selection of subspaces, while an efficient tabu search has been applied to each selected subspace. Four test problems have been analyzed and the results have been attractive, showing that the proposed methodology can be used as an efficient alternative to GA. When compared to GA, SP/TG results in a superior performance in terms of solution quality, execution time and reduced variability. By re-implementing the GA, we have not only shown the greater efficiency of our SP/TG but also reproven that the GA proposed in [50] is effective for the non-homogeneous RAP of series-parallel MSS.

$A_0 = 0.900$



$A_0 = 0.960$



$A_0 = 0.990$

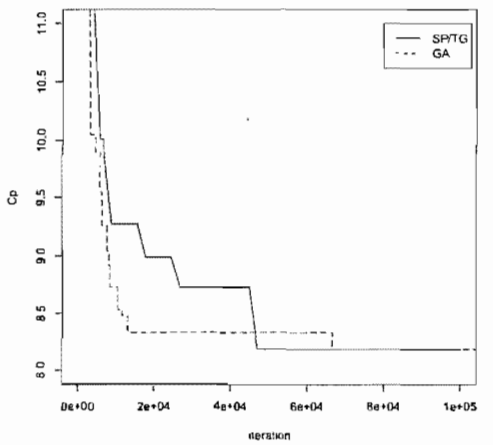
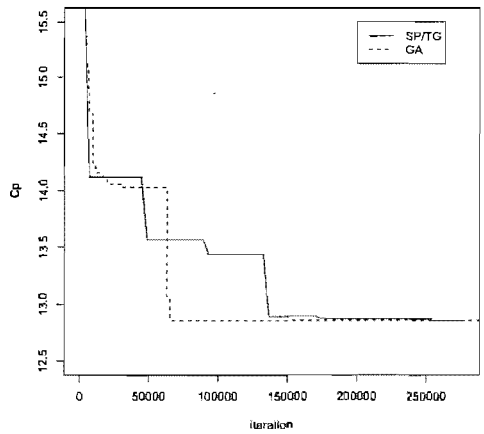
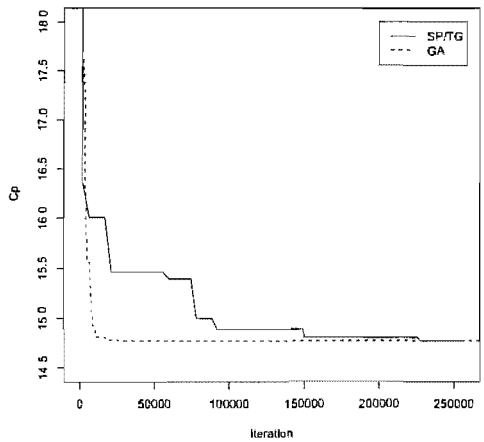


FIG. 3.1 – Convergence curves for GA and SP/TG for lev4-(4/6)-3

$A_0 = 0.975$



$A_0 = 0.980$



$A_0 = 0.990$

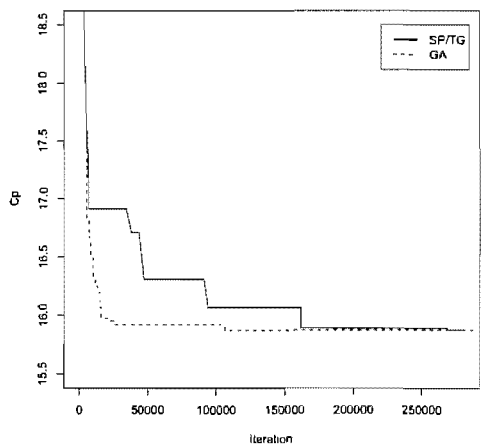
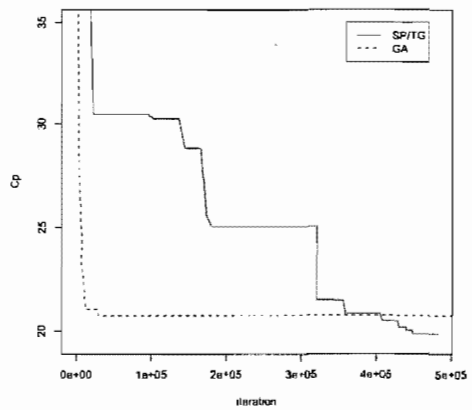
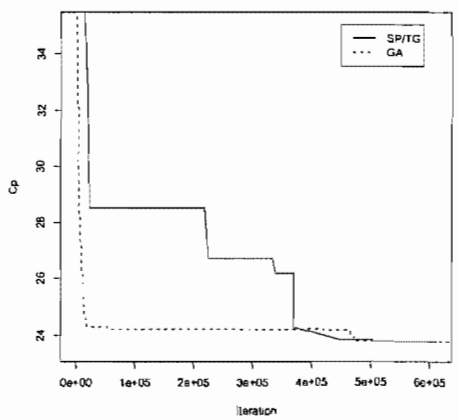


FIG. 3.2 – Convergence curves for GA and SP/TG for lev5-(4/9)-4

$A_0 = 0.950$



$A_0 = 0.990$



$A_0 = 0.999$

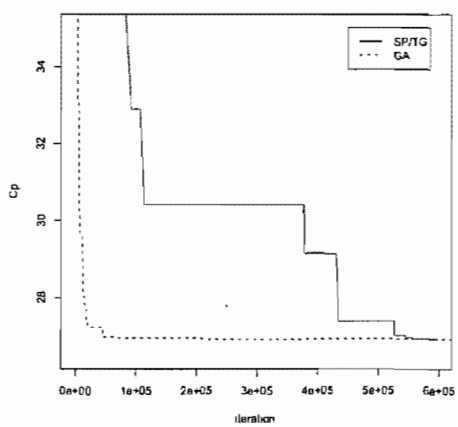
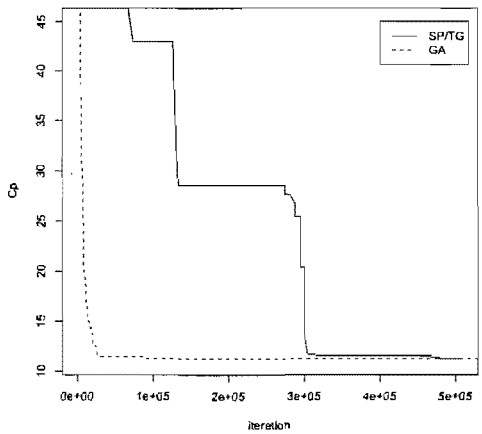
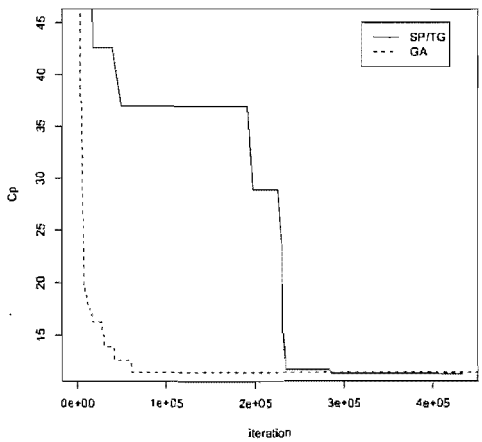


FIG. 3.3 – Convergence curves for GA and SP/TG for lis4-(7/11)-4

$A_0 = 0.975$



$A_0 = 0.980$



$A_0 = 0.990$

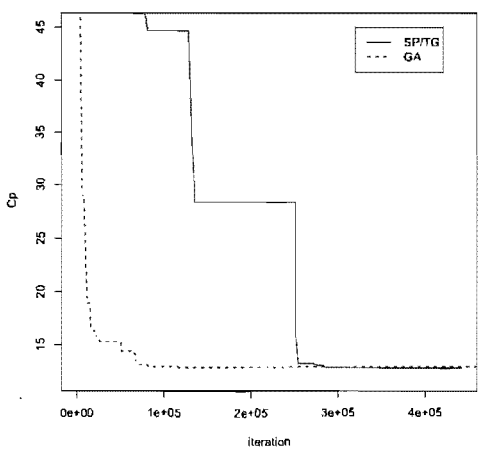


FIG. 3.4 – Convergence curves for GA and SP/TG for ouz6-(4/11)-4

Chapter 4

An efficient heuristic for reliability design optimization problems

Résumé. Nous proposons dans cet article une approche de résolution effectuée pour deux types différents de problèmes de conception optimale des systèmes :

1. Le problème d'allocation de la redondance (PAR) des systèmes binaires. On considère que le système et ses composantes ne peuvent occuper que deux états : fonctionnement parfait et défaillance complète. L'objectif est de maximiser la fiabilité du système série-parallèle sous des contraintes de coût et de poids. Ce problème est "difficile" à résoudre compte tenu du grand nombre de solutions possible. En fait, l'analyse de la complexité théorique de calcul a montré que ce problème était NP-difficile [14], ce qui implique que la résolution exacte d'instances de taille substantielle, susceptibles d'être rencontrées dans des applications réelles, demande en général des temps de calcul considérables. Les résultats numériques obtenus pour 33 problèmes tests, existants dans la littérature du PAR, montrent que l'algorithme proposé est très compétitif par rapport aux autres approches publiées dans la littérature.

2. Le problème de la planification des extensions multi-périodes pour les systèmes multi-états (SME) série-parallèles. Ce problème est une extension du problème d'allocation de la redondance dans un système multi-états série-parallèle. On considère que chaque composante peut occuper plusieurs états qui ont des niveaux de performance différents. La disponibilité du SME sera mesurée par la probabilité que le système satisfasse la demande de consommateur (performance requise). L'horizon d'étude est divisé en plusieurs périodes. À chaque période, des composantes sont incluses afin de réaliser un niveau souhaitable de disponibilité de production. Les composantes ajoutées dans le système servent pour toutes les périodes à partir de la période où elles ont été ajoutées. Chaque composante peut être ajoutée dans n'importe quel sous-système et à n'importe quelle période. Ces composantes sont caractérisées par leurs coûts, leurs performances et leurs disponibilités; elles sont choisies parmi une liste de produits disponibles dans le marché. L'objectif est de minimiser la somme des coûts d'investissement sur l'ensemble de l'horizon d'étude, tout en satisfaisant la contrainte de disponibilité à chaque période. Les résultats numériques obtenus pour 6 problèmes tests issus de la littérature montrent l'efficacité de l'algorithme développé par rapport aux meilleurs résultats publiés dans la littérature.

An efficient heuristic for reliability design optimization problems

Mohamed OUZINEB
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT),
Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Mustapha NOURELFATH
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT),
Département de génie mécanique,
Université Laval, Local 3344, Pavillon Adrien-Pouliot Sainte-Foy,
Québec G1K 7P4, Canada.

Michel GENDREAU
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT),
Département d'informatique et de recherche opérationnelle,
Université de Montréal, Case postale 6128, succursale "Centre-ville",
Montréal H3C 3J7, Canada.

Article a été accepté en 2009 pour publication dans le journal *Computers and Operations Research.*

Mai 2009

Abstract. This paper develops an efficient heuristic to solve two typical combinatorial optimization problems frequently met when designing highly reliable systems. The first one is the redundancy allocation problem (RAP) of series-parallel binary-state systems. The design goal of the RAP is to select the optimal combination of elements and redundancy levels to maximize system reliability subject to the system budget and to the system weight. The second problem is the expansion-scheduling problem (ESP) of multi-state series-parallel systems. In this problem, the study period is divided into several stages. At each stage, the demand is represented as a piecewise cumulative load curve. During the system lifetime, the demand can increase and the total productivity may become insufficient to assume the demand. To increase the total system productivity, elements are added to the existing system. The objective in the ESP is to minimize the sum of costs of the investments over the study period while satisfying availability constraints at each stage. The heuristic approach developed to solve the RAP and the ESP is based on a combination of space partitioning, genetic algorithms (GA) and tabu search (TS). After dividing the search space into a set of disjoint subsets, this approach uses GA to select the subspaces, and applies TS to each selected subspace. Numerical results for the test problems from previous research are reported and compared. The results show the advantages of the proposed approach for solving both problems.

Keywords. Tabu search, Genetic algorithms, Space partitioning, Redundancy optimization, Expansion planning

4.1 Introduction

During the last decades, reliability optimization has attracted a great number of researchers, due to the critical importance of designing highly reliable systems in various industrial contexts. One option to improve system reliability or availability is to include redundant elements in parallel. Nevertheless, resources are usually required for any enhancement leading to constrained optimization problems. In general these problems are nonlinear integer programming problems, of combinatorial nature and NP-hard. For example, the redundancy allocation problem (RAP) was shown to be NP-hard in [14]. In recent works a major focus is on the development of heuristic methods that are based on meta-heuristics to solve reliability optimization problems [43].

The present contribution develops an efficient approach based on meta-heuristics to solve two problems of system reliability optimization, namely the redundancy allocation problem (RAP) of series-parallel binary-state systems (Problem 1), and the expansion-scheduling problem (ESP) of series-parallel multi-state systems (Problem 2). Problems 1 and 2 are chosen as typical representatives from binary-state reliability and multi-state reliability problems, respectively. The proposed heuristic approach combines genetic algorithms (GA), tabu search (TS), and the idea of space partitioning (SP). Because of such a combination, it is said to be hybrid. We call it space partitioning/tabu-genetic (SP/TG), SP and TG being acronyms of Space Partitioning and Tabu-Genetic, respectively.

The remainder of the paper is organized as follows. In Section 2, we present briefly a description, a literature review, and a mathematical formulation for problems 1 and 2. Section 3 develops our SP/TG approach. In Section 4, we apply this approach to solve problems 1 and 2. The test problems and the numerical results are presented in Section 5. Finally, some concluding remarks are given in Section 6.

4.2 Two typical reliability design optimization problems

4.2.1 The redundancy allocation in series-parallel binary-state systems

4.2.1.1 Problem description and literature review

Using binary-state reliability modeling, Problem 1 assumes that a system and its elements may experience only two possible states: good and failed. We also assume that failed elements are not repaired. Other classical assumptions include that all redundancy is active, and that the failures of individual elements are mutually independent. The system consists of components in series. Each component contains a number of elements connected in parallel. Different elements can be placed in parallel (*i.e.*, element mixing is allowed). A component of index i is functioning correctly if at least k_i of its elements are operational (k-out-of n:G component redundancy). This series-parallel system is a logic diagram representation. For each component, there are various element versions, which are proposed by the suppliers in the market. Each element is characterized by its cost, its weight and its reliability according to its version. Redundancy allows reliability improvement, but increases the total cost. The design goal is to select the optimal combination of elements and redundancy levels so that the total reliability is maximized, subject to budget and to weight constraints. This is a complex combinatorial optimization problem, which is very important in many industrial applications. It has been studied in many different forms, and by considering numerous approaches and techniques.

Assuming element mixing is not allowed, the RAP of binary-state series-parallel systems has been solved using exact methods (in [8, 20] using dynamic programming approach, in [12, 21, 27, 59] using integer programming and in [77] using mixed-integer and nonlinear programming). In [63], Nakagawa and Miyazaki have also proposed an exact algorithm named N&M to solve 33 variations of the Fyffe problem [20]; they have shown that their algorithm found optimal solutions for 30 of them. But, the algorithm

did not converge to a feasible solution for the 3 other cases.

When element mixing is permitted, the problem becomes very complex due to the enormous size of the search space. To solve the problem, Coit and Smith [16] used a genetic algorithm while allowing to search in infeasible regions. A linear approximation for RAP with multiple element choices has been suggested by Hsieh [35]. An efficient tabu search has been developed in [41] to solve the RAP, while the ant colony optimization meta-heuristic was used in [54]. To the best of our knowledge, the best-published results have been provided by ant colony meta-heuristic with the degraded ceiling in [62] and by variable neighbourhood search in [53].

4.2.1.2 Mathematical formulation

Notation for Problem 1

n	number of components in the system
i	index for component, $i \in \{1, 2, ..., n\}$
m_i	number of available elements choices for component i
k_i	minimum number of elements in parallel required for component i
$\mathbf{k}$	$(k_1, k_2, ..., k_n)$
p_i	total number of elements used in component i
n_{max}	maximum number of elements in parallel
j	index (type) for element
x_{ij}	number of element of type j used in component i
$\mathbf{x}_i$	$(x_{i1}, x_{i2}, ..., x_{im_i})$
$\mathbf{X}$	a string of dimension $L = \sum_{i=1}^{n} m_i$ which defines the entire system structure, $\mathbf{X} = (x_{11}, x_{12}, ..., x_{1m_1}, x_{21}, x_{22}, ..., x_{2m_2}, ..., x_{n1}, x_{n2}, ..., x_{nm_n})$
M_{ij}	maximum number of elements of version j in parallel belonging to component i
r_{ij}	reliability of element of version j belonging to component i
c_{ij}	cost of each element of version j in component i

w_{ij}	weight of each element of version j in component i
$R(\mathbf{x}_i\|k_i)$	total reliability of system, given k_i
$R_i(\mathbf{x}_i\|k_i)$	reliability of component i, given k_i
$C_i(\mathbf{x}_i)$	total cost of component i
$W_i(\mathbf{x}_i)$	total weight of component i
C_0	cost limit
W_0	weight limit
$mnli$	maximum number of local iterations without improvement

The RAP formulation is given by:

$$\text{maximize} \quad R(\mathbf{x}_i|k_i) = \prod_{i=1}^{n} R_i(\mathbf{x}_i|k_i) \tag{4.1}$$

subject to

$$\sum_{i=1}^{n} C_i(\mathbf{x}_i) \leq C_0, \tag{4.2}$$

$$\sum_{i=1}^{n} W_i(\mathbf{x}_i) \leq W_0, \tag{4.3}$$

$$\sum_{j=1}^{m_i} x_{ij} \leq n_{max}, \quad i = 1, 2, ..., n, \tag{4.4}$$

$$\sum_{j=1}^{m_i} x_{ij} \geq k_i, \qquad i = 1, 2, ..., n, \tag{4.5}$$

$$x_{ij} \in \{0, 1, ..., M_{ij}\}, \quad i = 1, 2, ..., n; \quad j = 1, 2, ..., m_i. \tag{4.6}$$

The objective function (4.1) represents overall reliability of the series-parallel system. The system reliability is calculated by the product of n component reliabilities represented by $R_i(\mathbf{x}_i|k_i)$. The value $R_i(\mathbf{x}_i|k_i)$ is the reliability of component i given the minimum number of elements in parallel required for component i to function, *i.e.*, k_i.

Constraints (4.2) and (4.3) represent, respectively, the budget and the weight constraints. Constraint (4.4) specifies that, the number of elements to be included into component i cannot be higher than a pre-selected maximum number n_{max}. Constraint (4.5) represents the k-out-of n:G constraint. Constraint (4.6) specifies that, for each

component i, number of elements of each type j is identified by integers from 0 to a maximum number of elements available in the market.

4.2.2 The expansion-scheduling of series-parallel multi-state systems

4.2.2.1 Problem description and literature review

The expansion-scheduling problem (ESP) of a series-parallel multi-state system was introduced by Levitin in [45]. The system is called a multi-state system (MSS) because it is considered to have a range of performance levels from perfect functioning to complete failure. In the formulation of the ESP of a series-parallel MSS, the system study horizon is divided into several periods. At each period the demand distribution is predicted in the form of a cumulative demand curve. As in many industrial cases the demand increases with time, the design problem concerns expansions (or reinforcements) to adjust the system productivity (or capacity) to the demand. This is ensured by adding elements which are chosen from the list of products available in the market. It is assumed that any failed element is repaired. These elements are binary-state and are characterized by their cost, productivity and own availability. The MSS availability is defined as the ability to satisfy consumer demand. The initial structure of the system may be given at the initial stage or it may be empty. The objective is to minimize the sum of investment-cost over the study period while satisfying availability constraints at each stage. The developed algorithm has to answer the following questions:

1. What elements must be added to the system?
2. Into which system-components the additional elements should be included?
3. At which period stage each element should be added?

The ESP is a difficult large-scale optimization problem due to the enormous size of the search space. It can be seen as a more complicated extension of the RAP. In fact, the single stage series-parallel MSS expansion-scheduling problem corresponds to the

RAP of MSS. A good and extensive review of MSS literature can be found for example in [46] and [55]. The RAP for series-parallel MSS was first introduced in [81], where the universal moment generating function (UMGF) method was used for the reliability calculation [80]. Following these works, genetic algorithms were used in [50] and [51] for finding the minimal cost configuration of a series-parallel MSS under reliability or availability constraints. The others existing solution techniques of the RAP of MSS are ant colony optimization [68], a heuristic algorithm in [73] and tabu search [71]. The expansion-scheduling (or expansion planning) problem for multi-state series-parallel systems is more recent and has not been sufficiently studied. The only existing solution techniques of this problem are genetic algorithms [45] and an ant colony optimization algorithm in [68].

4.2.2.2 Mathematical formulation

Notation for Problem 2

Γ	number of expansion stages
γ	stage number, $1 \leq \gamma \leq \Gamma$
$J_{\gamma i}$	a number which identifies version (version number or index) of element to be included into component i at stage γ, $J_{\gamma i} \in \{1, 2, ..., Max(J_{\gamma i})\}$
$Max(J_{\gamma i})$	maximum number of versions available in the market for element to be included into component i at stage γ
$\mathbf{J}$	a matrix $(J_{\gamma i})$, $1 \leq \gamma \leq \Gamma$, $1 \leq i \leq n$, which specifies version numbers to be included to each component at each stage in the system
$H_{\gamma i}$	number of elements to be included into component i at stage γ
$Max(H_{\gamma i})$	maximum $H_{\gamma i}$ allowed (i.e., the upper bound of $H_{\gamma i}$)
$\mathbf{H}$	a matrix $(H_{\gamma i})$, $1 \leq \gamma \leq \Gamma$, $1 \leq i \leq n$, which specifies the numbers of elements to be included into each component in the system at each stage γ, $H_{\gamma i} \in \{1, 2, ..., Max(H_{\gamma i})\}$

$\mathbf{Y}$	a matrix $(Y_{\gamma l})$, $1 \leq \gamma \leq \Gamma$, $1 \leq l \leq 2n$ of dimension $2n\gamma$, which defines the entire expansion planning. For $1 \leq l \leq n$, the components of the matrix $\mathbf{Y}$ are those of $\mathbf{H}$. For $n+1 \leq l \leq 2n$, the components of the matrix $\mathbf{Y}$ are those of $\mathbf{J}$. This matrix is denoted by $\mathbf{Y} = (\mathbf{H}, \mathbf{J})$
$C_\gamma(\mathbf{Y})$	system expansion cost at stage γ, $C_\gamma(\mathbf{Y}) = C_\gamma(\mathbf{H}, \mathbf{J})$
$C(\mathbf{Y})$	total system expansion cost, $C(\mathbf{Y}) = \sum_{\gamma=1}^{\Gamma} C_\gamma(\mathbf{Y})$
$A_\gamma(\mathbf{Y})$	stationary availability index of the overall multi-state series-parallel system at stage γ
A_0	a specified minimum required level of system availability index
ρ	interest rate
$t(\gamma)$	time (in years) from initial stage to stage γ
j	index for element version ($j = J_{\gamma i}$)
A_{ij}	binary-state availability of element of version j belonging to component i
C_{ij}	cost of each element of version j in component i
W_{ij}	nominal performance level of element of version j in component i
A_{ij}^0	binary-state availability of element of version j belonging to i at stage 0
W_{ij}^0	nominal performance level of element of version j in component i at stage 0
K	number of partitioned intervals at each stage
k	index for partitioned intervals
$T(\gamma)$	MSS operation period at stage γ
$T_k(\gamma)$	a partitioned interval in $T(\gamma)$, $T(\gamma) = \sum_{k=1}^{K} T_k(\gamma)$
$W_k(\gamma)$	required MSS performance level for $T_k(\gamma)$
m	number of MSS state, $m \in \{1, 2, ..., M\}$, 1 is the worst state and M is the best state
W_m	MSS steady-state performance level associated with m
$W(t)$	output performance level of the MSS at time t, $W(t) \in \{W_1, ..., W_M\}$
p_{rm}	$\lim_{t \to \infty}[Pr(W(t) = W_m)]$
Str_0	initial system structure
q	amplification parameter in the penalized objective function

The global period to be examined is divided into several stages. Each stage γ begins $t(\gamma)$ years after the stage 0. The total operation period at each stage is divided into K intervals of durations $(T_1(\gamma), T_2(\gamma), ..., T_K(\gamma))$, $(\gamma = 1, 2, ..., \Gamma)$ ($T_k(\gamma) = 0$ for redundant intervals), and each interval has a required demand level $(W_1(\gamma), W_2(\gamma), ..., W_K(\gamma))$, $(\gamma = 1, 2, ..., \Gamma)$. We consider an initial structure of the series-parallel system which consists of n components which are independent and connected in series. Each component i $(i = 1, 2, ..., n)$ is composed of actively redundant elements connected in parallel. Each element is characterized by its availability, its unit cost and its performance. Redundancy allows availability improvement, but increases the total cost. The objective is to minimize the sum of the costs of the investments over the study period, subject to availability constraints at each stage. At each stage, it is considered that once an element selection is made, only the same element type can be used to provide redundancy. That is, at each stage, for each component, one has to select one element type to be included and to determine the number of redundant elements. Figures 4.1–4.3 illustrate the expansion process in the ESP by considering an example. In reference [45], the ESP was studied for a power-station coal transportation system which supplies boilers, and which has 5 basic system-components.

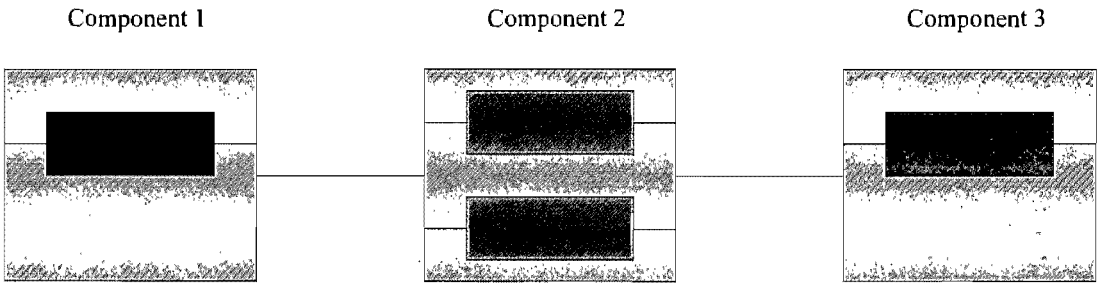


FIG. 4.1 – Initial structure of MSS

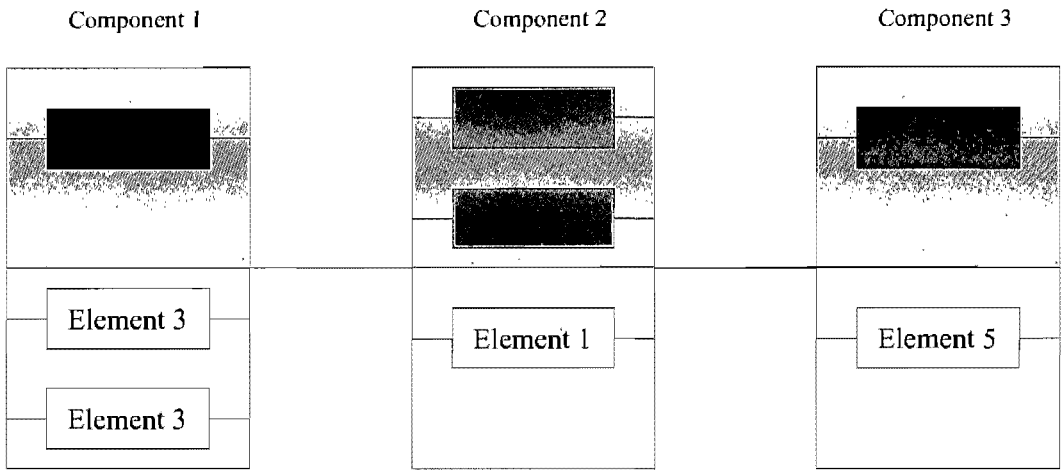


FIG. 4.2 – MSS structure for the first period

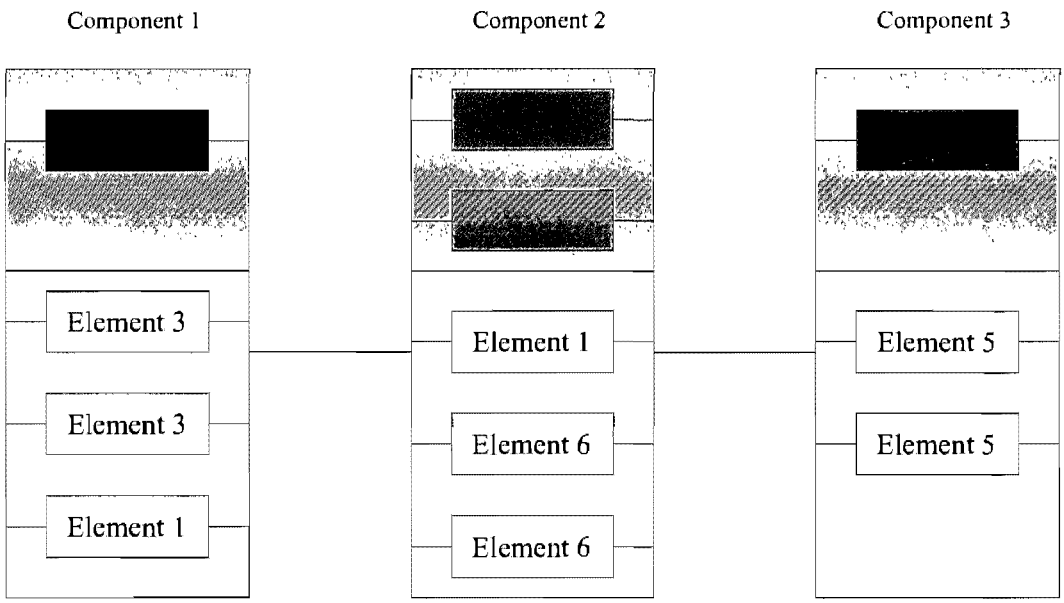


FIG. 4.3 – MSS structure for the second period

The total cost is calculated by adding the costs of the system expansion at each stage. For each system structure defined as a matrix $\mathbf{Y} = (Y_{\gamma l})$, $1 \leq \gamma \leq \Gamma$, $1 \leq l \leq 2n$, such as $\mathbf{Y}$=$(\mathbf{H}, \mathbf{J})$, the cost of the system expansion at stage γ is given by [45]:

$$C_\gamma(\mathbf{Y}) = \frac{1}{(1+\rho)^{t(\gamma)}} \sum_{i=1}^{n} H_{\gamma i} C_{i J_{\gamma i}}. \tag{4.7}$$

Thus, the total cost is given by:

$$C(\mathbf{Y}) = \sum_{\gamma=1}^{\Gamma} \frac{1}{(1+\rho)^{t(\gamma)}} \sum_{i=1}^{n} H_{\gamma i} C_{iJ_{\gamma i}}. \tag{4.8}$$

The availability of total system at stage γ is given by [45]:

$$A_\gamma(\mathbf{Y}) = \frac{\sum_{k=1}^{K} P(W_S(\gamma) \geq W_k(\gamma)) T_k(\gamma)}{\sum_{k=1}^{K} T_k(\gamma)}, \tag{4.9}$$

where, $P(W_S(\gamma) \geq W_k(\gamma))$ is the probability that the total system capacity $W_S(\gamma)$ at stage γ is not less than the demand level $W_k(\gamma)$.

The problem formulation is given by:

$$\text{minimize } C(\mathbf{Y}) = C(\mathbf{H}, \mathbf{J}) = \sum_{\gamma=1}^{\Gamma} \frac{1}{(1+\rho)^{t(\gamma)}} \sum_{i=1}^{n} H_{\gamma i} C_{iJ_{\gamma i}} \tag{4.10}$$

subject to

$$A_\gamma(\mathbf{H}, \mathbf{J}) \geq A_0, \quad \gamma = 1, 2, ..., \Gamma, \tag{4.11}$$

$$H_{\gamma i} \in \{0, 1, ..., Max(H_{\gamma i})\}, \quad i = 1, 2, ..., n; \quad \gamma = 1, 2, ..., \Gamma, \tag{4.12}$$

$$J_{\gamma i} \in \{1, ..., Max(J_{\gamma i})\}, \quad i = 1, 2, ..., n; \quad \gamma = 1, 2, ..., \Gamma. \tag{4.13}$$

The objective (4.10) is to minimize the total cost. Eq. (4.11) represents the availability constraint. Constraint (4.12) specifies that, at each stage γ, the number of elements to be included into component i is an integer which cannot be higher than a pre-selected maximum number $Max(H_{\gamma i})$. Constraint (4.13) specifies that, at each stage γ, for each component i, versions are identified by integers from 1 to a maximum number of versions available in the market. Given the solution structure ($\mathbf{Y} = (\mathbf{H}, \mathbf{J})$), the identical elements constraint is verified automatically. To compute the availability function $A_\gamma(\mathbf{H}, \mathbf{J})$ we will use the U-function (universal z-transform) technique [80]. This mathematical technique is also called universal moment generating function (UMGF). It was proven to be very effective for high dimension combinatorial problems in [46] and [55]. For more details about MSS availability evaluation by using the UMGF method, the reader is referred to [80] or [55]. This method is used in [45] to calculate the availability function $A_\gamma(\mathbf{H}, \mathbf{J})$.

4.2.2.3 Availability of repairable Multi-State Systems

The U-function for a discrete random variable $W_S(\gamma)$ is defined as a polynomial:

$$U_\gamma(z) = \sum_{m=1}^{M_\gamma} p_m(\gamma) z^{W_m(\gamma)}, \tag{4.14}$$

where

$p_m(\gamma) = P[W_S(\gamma) = W_m(\gamma)]$ and $W_S(\gamma)$ has M_γ possible values.

As in [45], $P[W_S(\gamma) \geq W_k(\gamma)]$ can be obtained using the δ operator:

$$\begin{aligned} P[W_S(\gamma) \geq W_k(\gamma)] &= \delta[U_\gamma(z), W_k(\gamma)] \\ &= \sum_{W_m(\gamma) \geq W_k(\gamma)} p_m(\gamma). \end{aligned} \tag{4.15}$$

The essential property of the U-function allows the total UMGF for a system of elements connected in parallel or in series to be obtained using simple algebraic operations over the individual U-functions of elements. To perform such operations, any used operator ψ has to satisfy the following Ushakov's four properties [80]:

1. $\psi(pz^a) = pz^a$.
2. $\psi(p_1 z^{a_1}, p_2 z^{a_2}) = p_1 p_2 z^{f(a_1,a_2)}$, where $f(a_1, a_2)$ is defined according to the physical nature of the MSS performance and the interactions between MSS elements:
   - $f(a_1, a_2) = \min(a_1, a_2)$ if elements are connected in series.
   - $f(a_1, a_2) = a_1 + a_2$ if elements are connected in parallel.
3. $\psi(U_1, ..., U_n) = \psi(\psi(U_1, ..., U_k), \psi(U_{k+1}, ..., U_n))$ for any k.
4. $\psi(..., U_k, U_{k+1}, ...) = \psi(..., U_{k+1}, U_k, ...)$ for any k.

Assume that only elements with total failures are considered, we have:

$$P[W_S(\gamma) = W_{ij}] = A_{ij} \text{ and } P[W_S(\gamma) = 0] = 1 - A_{ij},$$

for each element of version j used in component i.

So, the U-function for component i at the initial stage (stage 0) is:

$$U_i^0(z) = (1 - A_{iJ_{0i}}^0)z^0 + A_{iJ_{0i}}^0 z^{W_{iJ_{0i}}^0}. \tag{4.16}$$

If the component i containing X_{0i} parallel connected elements, then according to Ushakov's properties with $f(a_1, a_2) = a_1 + a_2$ we have:

$$\begin{aligned} U_i^0(z) &= [(1 - A_{iJ_{0i}}^0)z^0 + A_{iJ_{0i}}^0 z^{W_{iJ_{0i}}^0}]^{X_{0i}} \\ &= \sum_{m=0}^{X_{0i}} \alpha_{im} z^{mW_{iJ_{0i}}^0}, \end{aligned} \tag{4.17}$$

where

$$\alpha_{im} = \left[\frac{X_{0i}!}{m!(X_{0i} - m)!}\right] (A_{iJ_{0i}}^0)^m (1 - A_{iJ_{0i}}^0)^{X_{0i}-m}. \tag{4.18}$$

The U-function for component i at the stage γ is given by:

$$\begin{aligned} U_i^\gamma(z) &= \left[\prod_{n=0}^{\gamma-1} U_i^n(z)\right] \\ &\quad \cdot \left[[(1 - A_{iJ_{\gamma i}})z^0 + A_{iJ_{\gamma i}} z^{W_{iJ_{\gamma i}}}]^{X_{\gamma i}}\right] \\ &= \sum_{m=0}^{M_\gamma} \delta_m(\gamma) z^{W_m(\gamma)}. \end{aligned} \tag{4.19}$$

Therefore, according to (4.15) and (4.19) we have:

$$P_i[W_S(\gamma) \geq W_k(\gamma)] = \sum_{W_m(\gamma) \geq W_k(\gamma)} \delta_m(\gamma), \tag{4.20}$$

since,

$$P[W_S(\gamma) \geq W_k(\gamma)] = \prod_{i=1}^{s} P_i[W_S(\gamma) \geq W_k(\gamma)]. \tag{4.21}$$

Then according to (4.9) we have:

$$A_\gamma(Y) = \frac{\sum_{k=1}^{K} \prod_{i=1}^{s} \sum_{W_m(\gamma) \geq W_k(\gamma)} \delta_m(\gamma)}{\sum_{k=1}^{K} T_k(\gamma)}. \tag{4.22}$$

4.3 The space partitioning and tabu-genetic approach

The main idea of our approach consists in dividing the search space into a set of disjoint subsets, selecting subspaces by using genetic algorithms, and applying tabu search to each selected subspace. Each of these will now be described in more detail.

4.3.1 Dividing the search space into a set of disjoint subspaces

A given solution can be defined either as a vector or a matrix. It can be also defined as a string. We define an address which characterizes this solution. This address is usually an integer identifier which may depend on the system structure and parameters. It has to be chosen such that a set of solutions have the same address. In fact, we define each search subspace as the set of solutions which have the same address. The collection of non-empty search subspaces forms a *partition* of the search space.

4.3.2 Applying tabu search to each selected subspace

The role of this step is to look in each subspace for solutions by using an efficient TS. We define the neighbourhood structure. At each iteration of TS, the local transformations (or moves), that can be applied to the current solution, define a set of neighbouring solutions in a selected subspace as: *Neighbourhood(current solution)* = {*new solution* obtained by applying a single *move* to the *current solution*}. The move is chosen in such a way that the solution address does not change after a local transformation of the solution and the search process remains in the same subspace.

At each iteration, the best solution in a subset of *Neighbourhood(current solution)* is selected and considered as a tabu solution for some next iterations. This subset (referred to as the effective neighbourhood) is generated by eliminating the tabu solutions from *Neighbourhood(current solution).* Tabus are stored in a short-term memory of the search (tabu list). A previously visited solution is added to the tabu list in order to forbid the repetition of configurations. That is, tabus are used to prevent cycling when

moving away from local optima through non-improving moves. The size of the tabu list (denoted by length) is an important parameter regarding the efficiency of the heuristic, but its best value is not easy to be determined. A dynamic length is used, as it is usually found that it is more efficient to use a variable size tabu list [22, 71]. The termination criterion used can be specified in terms of a maximum number of local iterations without finding an improvement in the best-known solution.

As an important additional feature of our proposed TS, we use a penalty function while allowing infeasible solutions. This penalty function discourages, but allows, the TS algorithm to search into the infeasible boundary region. The idea of exploring around boundaries is known to be efficient for past implementations to solve the RAP of series-parallel systems. For example, the authors of [15] in their use of GA observed that better final feasible solutions could be found by permitting the exploration of the infeasible region, but by penalizing those solutions on the basis of the infeasibility degree. This idea is also used in [41, 54, 62] for solving the RAP of series-parallel binary-state systems with other meta-heuristics, in [61] for reliability optimization of a series system with multiple-choice and budget constraints, and in [71] for solving the RAP of series-parallel multi-state systems with TS. In TS, allowing infeasible solutions is a well-known idea [24, 28, 33]. An interesting way to find correct weights for constraint violations is to use self-adjusting penalties, *i.e.*, weights are adjusted dynamically on the basis of the recent history of the search. Weights are increased if only infeasible solutions were encountered in the last few iterations, and decreased if all recent solutions were feasible; see [24] for further details. Penalty weights can also be modified systematically to drive the search to cross the feasibility boundary of the search space and thus induce diversification. This technique, known as strategic oscillation, was first introduced in [28] and used since in several successful TS procedures. In the sequel, the TS algorithm applied to subspaces will be called TS-Sub.

4.3.3 Hybridization of genetic algorithm and tabu search

The proposed approach is based on a combination of space partitioning, genetic algorithms and tabu search. After dividing the search space into a set of disjoint subsets, this approach uses GA to select the subspaces, and applies TS to each selected subspace. This hybrid approach is called space partitioning/tabu-genetic (SP/TG), SP and TG being acronyms of Space Partitioning and Tabu-Genetic. When the number of subspaces is huge, the objective of this step is to locate promising search subspaces. SP/GA approach tends to provide a balance between diversification and intensification. First, the selection of subspaces by GA allows the identification of promising regions in the search space. One intended role of the used GA is to facilitate the exploration by guiding the search to unvisited subspaces. This leads to a diversification in subspaces. Second, the intended role of TS is to search carefully and intensively around good solutions found in the past search. This leads to intensification by exploiting each subspace selected by GA. Furthermore, as explained in the previous section, when using a penalty function while allowing infeasible solutions, we induce diversification by encouraging exploration of the boundary of the feasible region. Specifically, our algorithm has the same structure as GENITOR or "steady-state GA". With great respect to the genetic algorithm, mutation operator is replaced by the tabu search. GENITOR was developed in [84] and used in [51] and [45] for MSS reliability optimization. In general, GENITOR outperforms the basic "generational GA". The operators used in our algorithm can be summarized as follows (Figure 4.4):

1. **Encoding of solutions:** the great majority of the existing papers on GA use traditional binary encoding solutions. For combinatorial optimization problems, an encoding using integer values can be more efficient [16].

2. **Initial population:** the initial population was generated by randomly selecting solution vectors. Each individual in the population is represented as a string of finite integer numbers and represents only one region in the search space (*i.e.*, solutions which have the same address).

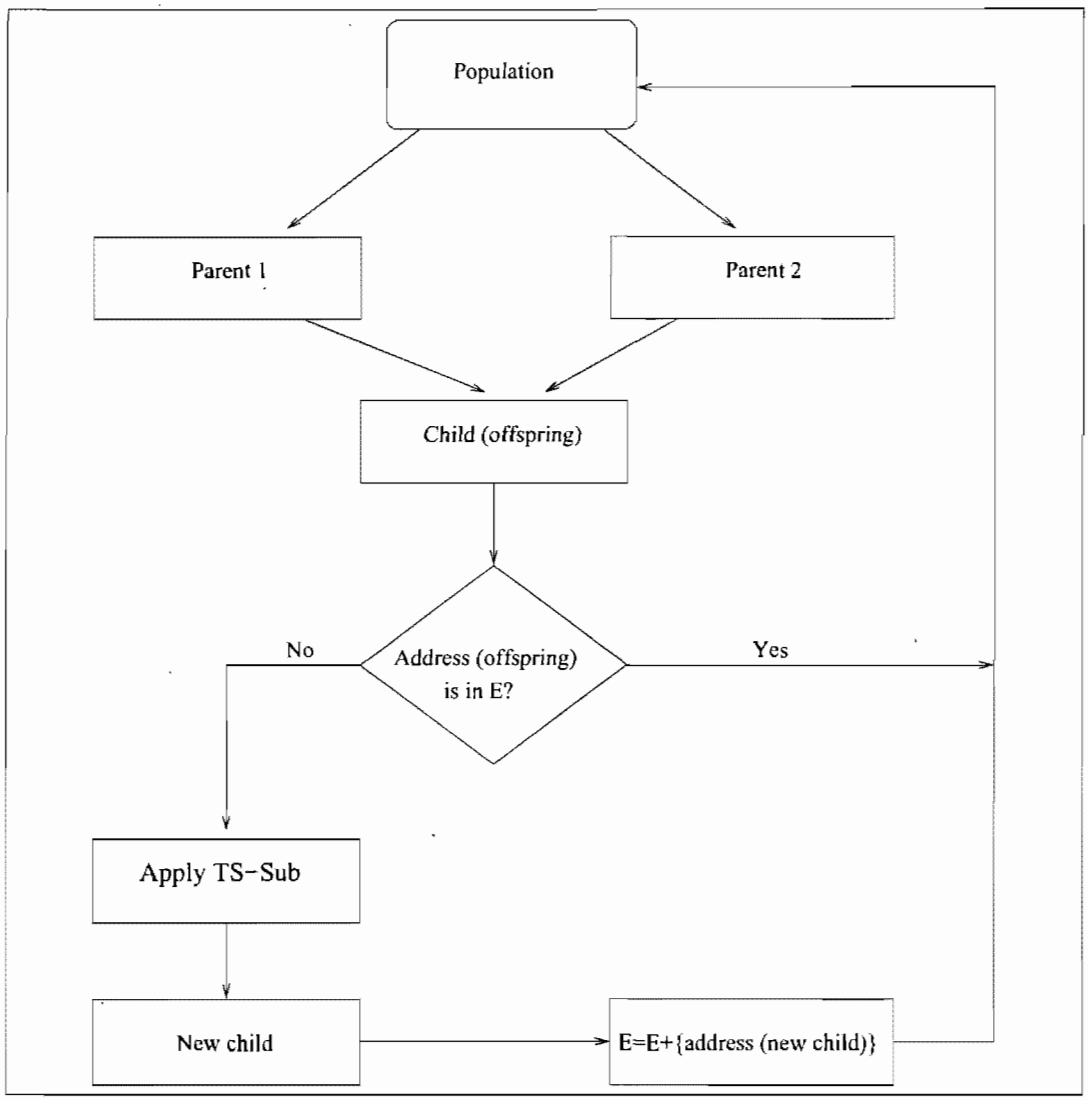


FIG. 4.4 – General algorithm

3. **Crossover breeding operator:** at each iteration, two individuals are randomly selected and produces a new solution (offspring) by combining the genetic material of two parents.

4. **Mutation operator (is replaced by TS):** this operator is used to find the best solution in the subspace represented by the new child using tabu search algorithm described above. The best solution found becomes the new child.

5. **Culling operator:** generates a new population. Each new solution obtained by TS is decoded to obtain the objective function (fitness) values. These val-

ues, which are a measure of quality, are used to compare different solutions.The comparison is performed by a selection procedure that specifies which solution is better: the new solution obtained by TS or the worst solution in the population. The better solution (representing the better subspace) joins the population, while the other is discarded. If the population contains equivalent subspaces following selection, redundancies are eliminated and the population size decreases consequently.

After new solutions are reproduced N_{rep} times (or if the population contains only one subspace), new randomly constructed solutions are generated to replenish the population, and a new genetic cycle begins. The GA is terminated after N_c genetic cycles. The final GA population contains the best selected subspace represented by its best solution found by TS. To avoid the application of TS to the same subspace more than once, a set E (initially empty) is used to store the addresses of already visited subspaces, while TS is not applied to the child if its address belongs to E.

4.4 Application of the proposed SP/TG approach

4.4.1 Application to the RAP

4.4.1.1 Dividing the search space into a set of disjoint subspaces

A given solution is defined by a system structure which is defined as a string $\mathbf{X}$ of dimension $L = \sum_{i=1}^{n} m_i$. The address of $\mathbf{X}$ is defined by:

$$address(\mathbf{X}) = \sum_{l=1}^{L} \mathbf{X}_l. \tag{4.23}$$

As a simple illustrative example, let consider a system of 2 components in series. Component 1 contains 2 elements of version 5 and 1 element of version 3 in parallel, and component 2 contains 3 elements of version 9 in parallel. The address of $\mathbf{X}$ is

simply obtained by summing the total number of elements (used in the system). That is, $address(\mathbf{X}) = 1 + 2 + 3 = 6$.

A search subspace of address r, denoted by S_r is defined as the set of solutions which have the same address, equal to r.

It follows from the above definitions that the lower bound of r is n and its upper bound is given by:

$$N = \sum_{i=1}^{n} \sum_{j=1}^{m_i} M_{ij}. \tag{4.24}$$

Note that $(S_r)_{n \leq r \leq N}$ is a partition of the search space S.

4.4.1.2 Applying tabu search to each selected subspace

In the previous subsection, the search space S has been partitioned into $(N - n + 1)$ subspaces $(S_n, S_{n+1}, ..., S_N)$. To apply TS to a given subspace S_{r_i} $(n \leq r_i \leq N)$, let define the neighbourhood structure. At each iteration of TS, the local transformations (or moves), that can be applied to the current solution $\mathbf{X}$, define a set of neighboring solutions in S_{r_i} as:
neighbourhood $(\mathbf{X})$ = {series-parallel structures obtained by applying a single move to $\mathbf{X}$}.

The move applied to $\mathbf{X}$ consists in changing the number of elements in parallel by adding and subtracting one, if possible, for any component. In this way, $address(\mathbf{X})$ does not change after a local transformation of $\mathbf{X}$ and the search process remains in the same subspace. To let the TS look for the solution with maximum total reliability and with $W \geq W_0$ and $C \geq C_0$, the fitness of a given individual is defined by:

$$F = \prod_{i=1}^{n} R_i(\mathbf{x}_i|k_i). \left(\frac{W}{W_0}\right)^{\alpha} . \left(\frac{C}{C_0}\right)^{\beta}. \tag{4.25}$$

By introducing a penalty function, we aim at encouraging the algorithm to explore the feasible region and infeasible region that is near the border of feasible area and discouraging, but allowing, search further into infeasible region.

4.4.1.3 Hybridization of genetic algorithm and tabu search

The operators used in our algorithm for the RAP are defined in Subsection 4.3.3. Encoding of solutions and Crossover breeding operator will be described in more detail:

1. **Encoding of solutions:** a given solution is defined by a system structure which is defined as a string of finite integer numbers ($\mathbf{X} = (x_{11}, x_{12}, ..., x_{1m_1}, x_{21}, x_{22}, ..., x_{2m_2}, ..., x_{n1}, x_{n2}, ..., x_{nm_n})$).

2. **Crossover breeding operator:** at each iteration, two individuals are randomly selected and produces a new solution (offspring) by combining the genetic material of two parents. The following 2-points crossover procedure is used:

   Given two parent strings X_1, X_2 and an offspring O, to define the offspring O, the following steps are used:

   (a) Generate randomly two positions k and m of the string O.

   (b) Copy X_2 in O between k and m positions.

   (c) Copy X_1 in O otherwise.

4.4.2 Application to the ESP

4.4.2.1 Dividing the search space into a set of disjoint subspaces

Given an expansion plan for a system as a matrix $\mathbf{Y} = (\mathbf{H}, \mathbf{J})$, ($\mathbf{Y}=(Y_{\gamma l})$, $1 \leq \gamma \leq \Gamma$, $1 \leq l \leq 2n$), the address of $\mathbf{Y}$ is defined by:

$$address(\mathbf{Y}) = \sum_{\gamma=1}^{\Gamma} \sum_{l=1}^{2n} Y_{\gamma l}. \tag{4.26}$$

For example, let consider the system such as the initial structure is empty, $n = 3$, $\Gamma = 3$,

$$\mathbf{H} = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \mathbf{J} = \begin{pmatrix} 3 & 1 & 5 \\ 1 & 6 & 5 \\ 2 & 1 & 3 \end{pmatrix} \text{ and } \mathbf{Y} = \begin{pmatrix} 2 & 1 & 1 & 3 & 1 & 5 \\ 1 & 0 & 1 & 1 & 6 & 5 \\ 1 & 0 & 0 & 2 & 1 & 3 \end{pmatrix}.$$

The address of $\mathbf{Y}$ is simply obtained by summing the total number of elements (added in the total system) with the sum of version numbers, *i.e.*, $address(\mathbf{Y}) = 34$.

A search subspace of address r, denoted by S_r is defined as the set of solutions which have the same address, equal to r.

It follows from the above definitions that the lower bound of r is 0 and its upper bound is given by:

$$N = \sum_{\gamma=1}^{\Gamma} \sum_{i=1}^{n} [Max(H_{\gamma i}) + Max(J_{\gamma i})]. \tag{4.27}$$

$(S_r)_{0 \leq r \leq N}$ is a partition of the search space S.

4.4.2.2 Applying tabu search to each selected subspace

In the previous subsection, the search space S has been partitioned into $(N+1)$ subspaces $(S_0, S_1, ..., S_N)$. To apply TS to a given subspace S_{r_i} $(0 \leq r_i \leq N)$, let us define the neighbourhood structure. At each iteration of TS, the local transformations (or moves), that can be applied to the current solution $\mathbf{Y}$, define a set of neighboring solutions in S_{r_i} as: neighbourhood($\mathbf{Y}$) = {expansion plan for a system obtained by applying a single move to $\mathbf{Y}$}.

The move applied to $\mathbf{Y}$ consists in changing the number and the version of elements by adding and subtracting one, if possible, for any component of the solution matrix $\mathbf{Y} = (\mathbf{H}, \mathbf{J})$, ($\mathbf{Y} = (Y_{\gamma l})$, $1 \leq \gamma \leq \Gamma$, $1 \leq l \leq 2n$). In this way, $address(\mathbf{Y})$ does not change after a local transformation of $\mathbf{Y}$ and the search process remains in the same subspace. Each single move can be applied in three different ways:

1. Changing the number of elements in components ($H_{\gamma_1 l_1} \rightarrow H_{\gamma_1 l_1} + 1$ and $H_{\gamma_2 l_2} \rightarrow H_{\gamma_2 l_2} - 1$);

2. Changing version numbers in components ($J_{\gamma_1 l_1} \rightarrow J_{\gamma_1 l_1} + 1$ and $J_{\gamma_2 l_2} \rightarrow J_{\gamma_2 l_2} - 1$);

3. Changing both number of elements and version numbers in components ($H_{\gamma_1 l_1} \rightarrow H_{\gamma_1 l_1} + 1$ and $J_{\gamma_2 l_2} \rightarrow J_{\gamma_2 l_2} - 1$) or ($H_{\gamma_1 l_1} \rightarrow H_{\gamma_1 l_1} - 1$ and $J_{\gamma_2 l_2} \rightarrow J_{\gamma_2 l_2} + 1$).

The fitness of a given individual is defined as follows [45]:

$$F = \sum_{\gamma=1}^{\Gamma} \frac{1}{(1+\rho)^{t(\gamma)}} \sum_{i=1}^{n} H_{\gamma i} C_{i J_{\gamma i}} + q \sum_{\gamma=1}^{\Gamma} \max\{0, A_0 - A_\gamma(\mathbf{H}, \mathbf{J})\}. \quad (4.28)$$

4.4.2.3 Hybridization of genetic algorithm and tabu search

The operators used in our algorithm for the ESP are defined in Subsection 4.3.3. Encoding of solutions and Crossover breeding operator will be described in more detail:

1. **Encoding of solutions:** a given solution is defined by a system structure which is defined as a matrix $\mathbf{Y} = (Y_{\gamma l})$, $1 \leq \gamma \leq \Gamma$, $1 \leq l \leq 2n$, such as $\mathbf{Y}$=($\mathbf{H}, \mathbf{J}$).

2. **Crossover breeding operator:** at each iteration two individuals are randomly selected and they produce a new solution (offspring) using a 2-point crossover procedure (defined below):

   Given two parent strings Y_1, Y_2 and an offspring O, to define the offspring O, the following steps are used:

   (a) Generate randomly two positions k and m of the string O.

   (b) Copy Y_2 in O between k and m positions.

   (c) Copy Y_1 in O otherwise.

   Figure 4.5 presents an example to illustrate this 2-point crossover procedure for $s = 3$, $\Gamma = 2$, $k = 4$ and $m = 8$.

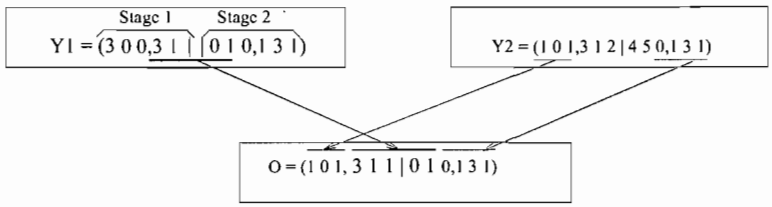


FIG. 4.5 – Example of 2-point crossover procedure

4.5 Computational results

For both problems, the algorithm was implemented in C^{++}. The numerical tests were completed on an Intel Pentium IV 3000 MHz DEC station 5000/240 with 1024 Mbytes of RAM running under Linux.

4.5.1 Redundancy allocation problem (RAP)

4.5.1.1 Test problems for the RAP

The test problems, used to evaluate the performance of the SP/TG algorithm when applied to the RAP, were originally proposed by Fyffe *et al.* in [20] and modified by Nakagawa and Miyazaki in [63]. Fyffe *et al.* [20] considered a system with 14 components and specified constraint limits of 130 units of system cost, 170 units of system weight and suppose 1-out-of n:G component redundancy. Nakagawa and Miyazaki [63] developed 33 variations of the Fyffe problem, where the cost constraint C is set to 130 and the weight constraint W is decreased incrementally from 191 units to 159 units. The element cost, weight and reliability values, as originally presented in [20], are reproduced in Table 4.1. For each component, there are three or four element choices.

Component i	Element choices											
	Choice 1			Choice 2			Choice 3			Choice 4		
	r_{i1}	c_{i1}	w_{i1}	r_{i2}	c_{i2}	w_{i2}	r_{i3}	c_{i3}	w_{i3}	r_{i4}	c_{i4}	w_{i4}
1	0.90	1	3	0.93	1	4	0.91	2	2	0.95	2	5
2	0.95	2	8	0.94	1	10	0.93	1	9	-	-	-
3	0.85	2	7	0.90	3	5	0.87	1	6	0.92	4	4

Component i	Element choices											
	Choice 1			Choice 2			Choice 3			Choice 4		
	r_{i1}	c_{i1}	w_{i1}	r_{i2}	c_{i2}	w_{i2}	r_{i3}	c_{i3}	w_{i3}	r_{i4}	c_{i4}	w_{i4}
4	0.83	3	5	0.87	4	6	0.85	5	4	-	-	-
5	0.94	2	4	0.93	2	3	0.95	3	5	-	-	-
6	0.99	3	5	0.98	3	4	0.97	2	5	0.96	2	4
7	0.91	4	7	0.92	4	8	0.94	5	9	-	-	-
8	0.81	3	4	0.90	5	7	0.91	6	6	-	-	-
9	0.97	2	8	0.99	3	9	0.96	4	7	0.91	3	8
10	0.83	4	6	0.85	4	5	0.90	5	6	-	-	-
11	0.94	3	5	0.95	4	6	0.96	5	6	-	-	-
12	0.79	2	4	0.82	3	5	0.85	4	6	0.90	5	7
13	0.98	2	5	0.99	3	5	0.97	2	6	-	-	-
14	0.90	4	6	0.92	4	7	0.95	5	6	0.99	6	9

TAB. 4.1: Data for RAP test problems [20]

4.5.1.2 Size of the search space and number of subspaces for the RAP

As in [53], the generation of initial solutions were controlled in a range between k_i and $M_{ij} - 4$ (inclusive). The total number of different solutions to be examined and the number of subspaces are simply given by the following equations:

$$\text{Size of the search space} = \prod_{i=1}^{n} \prod_{j=1}^{m_i} M_{ij}. \tag{4.29}$$

$$\text{Number of subspaces} = \sum_{i=1}^{n} \sum_{j=1}^{m_i} M_{ij} - n + 1. \tag{4.30}$$

Let us consider that different types of elements are allowed to reside in parallel, and assume that ($M_{ij} = 8 \ \forall j, \ 1 \leq j \leq m_i$, and $\forall i, \ 1 \leq i \leq n$). This means simply that a maximum number of 8 s-identical elements are allowed for each component. In this case, the search space size is $8^{49} \approx 2.10^{44}$, while the number of subspaces is 371.

4.5.1.3 Parameter settings for the RAP

Preliminary numerical tests were used to set the values of the parameters. Different data are randomly generated and used to calibrate the parameters. Once the values of the parameters are set for these preliminary data, they are used for the variations of the problem instances to be solved in this paper. In this way, we avoid any parameter overfitting. The parameters' values are: $N_s = 50$, $N_{rep} = 50$, $N_c = 10$, $mnli = 200$, $\alpha = 1$ and $\beta = 0.3$.

4.5.1.4 Comparing the best solutions of SP/TG and existing methods for the RAP

The performance of the SP/TG heuristic is compared with the best-known heuristics for the RAP from previous literature, namely the genetic algorithm (GA) in [16], the linear approximation (LA) approach in [35], the tabu search (TS) algorithm in [41], the ant colony optimization (ACO) in [54], the variable neighbourhood search (VNS) approach in [53], and the ant colony optimization coupled with the degraded ceiling algorithm (ACO/DC) in [62]. Ten runs of our algorithm were made using different random seeds for each problem instance. The best feasible solution over ten runs was first used for comparison. The results of the best solutions (*i.e.*, configuration, reliability, cost and weight) obtained by SP/TG for each of the 33 instances, are presented in Table 4.2. Table 4.3 gives a comparison between the best solutions of SP/TG and the best solutions obtained by the methods in references [16, 35, 41, 53, 54, 62]. Note that in these references, whenever the proposed algorithm is of a stochastic nature, 10 trials were performed and the best solution (among these 10 trials) was used as the final solution. The maximum reliability identified by these algorithms was then used to compare its performance to other algorithms. In Table 4.3, the best results are in boldface, indicating that generally SP/TG out-performed the existing approaches. More specifically, Table 4.3 shows that:

1. In 25 of the 33 test cases, the solutions found by our algorithm are better than

those found by the genetic algorithm in [16], while the rest (*i.e.,* 8 cases) are as good as those they found.

2. Our algorithm out-performed the linear approximation approach in Hsieh [35] for all instances.

3. In 7 of the 33 test cases, the solutions found by our algorithm are better than those found by the tabu search algorithm in [41], while the rest (*i.e.,* 26 cases) are as good those they found.

4. In 9 of the 33 test cases, the solutions found by our algorithm are better than those found by the ant colony optimization in [54], while the rest (*i.e.,* 24 cases) are as good as those they found.

5. Our algorithm outperformed the variable neighbourhood search approach proposed in [53] for 6 instances, while the rest (*i.e.,* 27 instances) are as good as those they found.

6. In 2 of the 33 test cases, the solutions found by our algorithm are better than those found by the ant colony optimization coupled with the degraded ceiling algorithm in [62], while the rest (*i.e.,* 31 cases) are the same as those they found.

No	Solution encoding ($\mathbf{X}$)	$C(\mathbf{X})$	$W(\mathbf{X})$	$R(\mathbf{X})$
1	333,11,444,3333,222,22,111,1111,12,233,33,1111,11,34	130	191	0.986811
2	333,11,444,3333,222,22,111,1111,11,233,33,1111,12,34	130	190	0.986416
3	333,11,444,3333,222,22,111,1111,23,233,13,1111,11,34	130	189	0.985922
4	333,11,444,3333,222,22,111,1111,23,223,13,1111,12,34	130	188	0.985378
5	333,11,444,3333,222,22,111,1111,13,223,13,1111,22,34	130	187	0.984688
6	333,11,444,333,222,22,111,1111,23,233,33,1111,22,34	129	186	0.984176
7	333,11,444,3333,222,22,111,1111,23,223,13,1111,22,33	130	185	0.983505
8	333,11,444,333,222,22,111,1111,33,233,33,1111,22,34	130	184	0.982994
9	333,11,444,333,222,22,111,1111,33,223,33,1111,22,34	129	183	0.982256
10	333,11,444,333,222,22,111,1111,33,333,33,1111,22,33	130	182	0.981518
11	333,11,444,333,222,22,111,1111,33,233,33,1111,22,33	129	181	0.981027
12	333,11,444,333,222,22,111,1111,33,223,33,1111,22,33	128	180	0.980290

No	Solution encoding ($\mathbf{X}$)	$C(\mathbf{X})$	$W(\mathbf{X})$	$R(\mathbf{X})$
13	333,11,444,333,222,22,111,1111,33,223,13,1111,22,33	126	179	0.979505
14	333,11,444,333,222,22,111,1111,33,222,13,1111,22,33	125	178	0.978400
15	333,11,444,333,222,22,111,113,33,223,13,1111,22,33	126	177	0.977596
16	333,11,444,333,222,22,33,1111,33,223,13,1111,22,33	124	176	0.976690
17	333,11,444,333,222,22,13,1111,33,223,33,1111,22,33	125	175	0.975708
18	333,11,444,333,222,22,13,1111,33,223,13,1111,22,33	123	174	0.974926
19	333,11,444,333,222,22,13,1111,33,222,13,1111,22,33	122	173	0.973827
20	333,11,444,333,222,22,13,113,33,223,13,1111,22,33	123	172	0.973027
21	333,11,444,333,222,22,13,113,33,222,13,1111,22,33	122	171	0.971929
22	333,11,444,333,222,22,13,113,33,222,11,1111,22,33	120	170	0.970760
23	333,11,444,333,222,22,11,113,33,222,13,1111,22,33	121	169	0.969291
24	333,11,444,333,222,22,11,113,33,222,11,1111,22,33	119	168	0.968125
25	333,11,444,333,22,22,13,113,33,222,11,1111,22,33	118	167	0.966335
26	333,11,44,333,222,22,13,113,33,222,11,1111,22,33	116	166	0.965042
27	333,11,444,333,222,22,11,113,33,222,11,1111,22,33	117	165	0.963712
28	333,11,44,333,222,22,11,113,33,222,11,1111,22,33	115	164	0.962422
29	333,11,44,333,22,22,13,113,33,222,11,1111,22,33	114	163	0.960642
30	333,11,44,333,22,22,11,113,33,222,13,1111,22,33	115	162	0.959188
31	333,11,44,333,22,22,11,113,33,222,11,1111,22,33	113	161	0.958035
32	333,11,44,333,22,22,11,111,33,222,13,1111,22,33	112	160	0.955714
33	333,11,44,333,22,22,11,111,33,222,11,1111,22,33	110	159	0.954565

TAB. 4.2: Results of the best solutions obtained by SP/TG

No	W_0	GA[1]	LA[2]	TS[3]	ACO[4]	VNS[5]	ACO/DC[6]	SP/TG
1	191	0.98675	0.986711	**0.986811**	0.986745	**0.98681**	**0.986811**	**0.986811**
2	190	0.98603	0.986316	**0.986416**	0.985905	**0.98642**	0.986316	**0.986416**
3	189	0.98556	0.985724	**0.985922**	0.985773	**0.98592**	**0.985922**	**0.985922**
4	188	0.98503	0.985031	**0.985378**	0.985329	0.98487	**0.985378**	**0.985378**
5	187	0.98429	0.984153	**0.984688**	**0.984688**	0.98467	**0.984688**	**0.984688**
6	186	0.98362	0.983879	**0.984176**	0.983801	**0.98418**	**0.984176**	**0.984176**
7	185	0.98311	0.983387	**0.983505**	**0.983505**	**0.98351**	**0.983505**	**0.983505**
8	184	0.98239	0.982204	**0.982994**	**0.982994**	**0.98299**	**0.982994**	**0.982994**

No	W_0	GA[1]	LA[2]	TS[3]	ACO[4]	VNS[5]	ACO/DC[6]	SP/TG
9	183	0.98190	0.981466	**0.982256**	0.982206	**0.98226**	0.982225	**0.982256**
10	182	0.98102	0.979690	**0.981518**	0.981468	0.98147	**0.981518**	**0.981518**
11	181	0.98006	0.979280	**0.981027**	0.980681	**0.98103**	**0.981027**	**0.981027**
12	180	0.97942	0.978327	**0.980290**	**0.980290**	**0.98029**	**0.980290**	**0.980290**
13	179	0.97906	0.978055	**0.979505**	**0.979505**	**0.97951**	**0.979505**	**0.979505**
14	178	0.97810	0.976878	**0.978400**	**0.978400**	0.97838	**0.978400**	**0.978400**
15	177	0.97715	0.975400	0.977474	**0.977596**	**0.97760**	**0.977596**	**0.977596**
16	176	0.97642	0.974975	**0.976690**	0.976494	**0.97669**	**0.976690**	**0.976690**
17	175	0.97552	0.973500	**0.975708**	**0.975708**	**0.97571**	**0.975708**	**0.975708**
18	174	0.97435	0.972328	0.974788	**0.974926**	**0.97493**	**0.974926**	**0.974926**
19	173	0.97362	0.970531	**0.973827**	**0.973827**	0.97381	**0.973827**	**0.973827**
20	172	0.97266	0.969232	**0.973027**	**0.973027**	**0.97303**	**0.973027**	**0.973027**
21	171	0.97186	0.967896	**0.971929**	**0.971929**	**0.97193**	**0.971929**	**0.971929**
22	170	**0.97076**	0.966776	**0.970760**	**0.970760**	**0.97076**	**0.970760**	**0.970760**
23	169	0.96922	0.965612	**0.969291**	**0.969291**	**0.96929**	**0.969291**	**0.969291**
24	168	**0.96813**	0.964150	**0.968125**	**0.968125**	**0.96813**	**0.968125**	**0.968125**
25	167	**0.96634**	0.962990	**0.966335**	**0.966335**	**0.96634**	**0.966335**	**0.966335**
26	166	**0.96504**	0.961210	**0.965042**	**0.965042**	**0.96504**	**0.965042**	**0.965042**
27	165	**0.96371**	0.959923	**0.963712**	**0.963712**	**0.96371**	**0.963712**	**0.963712**
28	164	**0.96242**	0.958601	**0.962422**	**0.962422**	**0.96242**	**0.962422**	**0.962422**
29	163	**0.96064**	0.957317	0.959980	**0.960642**	**0.96064**	**0.960642**	**0.960642**
30	162	0.95912	0.955547	0.958205	**0.959188**	**0.95919**	**0.959188**	**0.959188**
31	161	**0.95803**	0.954101	0.956922	**0.958034**	**0.95804**	**0.958034**	**0.958035**
32	160	0.95567	0.952953	0.955604	**0.955714**	0.95567	**0.955714**	**0.955714**
33	159	0.95432	0.950800	0.954325	**0.954564**	**0.95457**	**0.954564**	**0.954565**

TAB. 4.3: Comparison of the best solutions among heuristics

[1] GA genetic algorithm [16]

[2] LA linear approximation [35]

[3] TS tabu search [41]

[4] ACO ant colony optimization [54]

[5] VNS variable neighborhood search [53]

[6] ACO/DC ant colony optimization and degraded ceiling [62]

4.5.1.5 Robustness of the SP/TG algorithm for the RAP

To measure the robustness of the SP/TG algorithm, the standard deviations and the average reliability, over ten runs in each instance, are given in Table 4.4. We remark from this table that for each instance, the standard deviation is very low. This implies that the proposed method is robust. The low standard deviation of SP/TG can be interpreted as a sign of insensitivity to the initial solution and the random number seed.

4.5.1.6 Comparing the computational effort of SP/TG and existing methods for the RAP

When considering the computational effort, the proposed algorithm requires less number of iterations (*i.e.*, generated solutions, during the whole search process) than TS in [41] and requires a larger number of iterations than GA in [15, 16] and VNS in [53]. From the previous literature, the numbers of iterations given for the following methods are:

- the number of solutions generated in GA [15, 16] is 48,000 (a population size of 40 chromosomes and 1200 generations),
- TS in [41] evaluated an average of 350,000 solutions,
- the ACO algorithm in [54] needs about 100,000 evaluations or more,
- the ACO/DC approach [62] needs about 150,000 evaluations or more,

- the average number of evaluations used by VNS in [53] is around 120,000.

For each instance, the average numbers of solutions evaluated by SP/TG over ten runs are given in the last column of Table 4.4. The number of solutions evaluated by our SP/TG is approximately 231,645 on average (obtained by averaging the values in the last column of Table 4.4).

Furthermore, our approach is compared to the others with a similar amount of evaluations as follows: for each test case, we stop our algorithm after the number of evaluations needed for the other algorithm to reach its best solution, and we compare the solution of our algorithm over ten runs with this best solution. The results from this comparison are as follows:

- SP/TG solutions are better than GA solutions in 18 of the 33 test cases. GA solutions are better than SP/TG solutions in 9 of the 33 test cases.
- SP/TG solutions are better than ACO solutions in 7 of the 33 test cases. ACO solutions are better than SP/TG solutions in 6 of the 33 test cases.
- SP/TG solutions are better than VNS solutions in 3 of the 33 test cases. VNS solutions are better than SP/TG solutions in 5 of the 33 test cases.
- SP/TG solutions are better than ACO/DC solutions in 1 of the 33 test cases. ACO/DC solutions are better than SP/TG solutions in 3 of the 33 test cases.
- SP/TG solutions are better than TS solutions in 7 of the 33 test cases, while the rest are as good those they found.

No	W_0	Max R	Mean R	Std dev	Average evaluations
1	191	0.986811	0.986811	0.000000	327,749
2	190	0.986416	0.986416	0.000000	386,116
3	189	0.985922	0.985922	0.000000	227,884
4	188	0.985378	0.985327	0.000091	480,042

No	W_0	Max R	Mean R	Std dev	Average evaluations
5	187	0.984688	0.984593	0.000161	169,061
6	186	0.984176	0.983762	0.000764	213,086
7	185	0.983505	0.983465	0.000127	151,248
8	184	0.982994	0.982964	0.000009	133,120
9	183	0.982256	0.982081	0.000490	263,815
10	182	0.981518	0.981002	0.001124	285,481
11	181	0.981027	0.980425	0.000390	231,039
12	180	0.980290	0.979630	0.000892	205,258
13	179	0.979505	0.978908	0.000449	195,794
14	178	0.978400	0.978151	0.000403	203,054
15	177	0.977596	0.977596	0.000000	184,477
16	176	0.976690	0.976651	0.000083	247,013
17	175	0.975708	0.975562	0.000308	282,445
18	174	0.974926	0.974675	0.000528	207,254
19	173	0.973827	0.973461	0.000743	183,166
20	172	0.973027	0.972591	0.000702	183,513
21	171	0.971929	0.971191	0.000952	228,454
22	170	0.970760	0.970090	0.000549	193,096
23	169	0.969291	0.968919	0.000743	170,768
24	168	0.968125	0.967766	0.001137	216,559
25	167	0.966335	0.965544	0.002500	286,547
26	166	0.965042	0.964826	0.000452	253,756
27	165	0.963712	0.963712	0.000000	216,990
28	164	0.962422	0.962422	0.000000	179,041
29	163	0.960642	0.960642	0.000000	303,201
30	162	0.959188	0.959138	0.000106	190,968
31	161	0.958035	0.958035	0.000000	242,367
32	160	0.955714	0.955714	0.000000	211,747

No	W_0	Max R	Mean R	Std dev	Average evaluations
33	159	0.954565	0.954565	0.000000	190,198

TAB. 4.4: Maximum reliability, average reliability, standard deviation and average evaluation of the SP/TG algorithm

4.5.2 Expansion-scheduling problem (ESP)

4.5.2.1 Test problems for the ESP

The test problems, used to evaluate the performance of the SP/TG algorithm when applied to the ESP, were proposed by Levitin in [45]. The system to be optimized is a power-station coal transportation system which supplies boilers, and which has 5 basic components connected in series. Each component may contain a set of parallel elements. For each component, different elements types are available. Each element is considered as a binary-state unit, while the system is a MSS with series-parallel structure. Two cases are considered:

- The case where the initial structure-set is empty ($Str_0 = \emptyset$). In this case, the problem is to determine the initial system structure as well as its expansion plan.
- The case where the initial system structure exists at stage 0 ($Str_0 = Str_0^*$). In this case, the problem is to determine the system expansion plan.

The reader is referred to reference [45] for all the input data including:

- The characteristics of available elements, *i.e.*, cost, availability and nominal capacity values.
- The data of the "piecewise cumulative boiler system demand curves" at 5 stages and times from the present to the beginning of these future stages.

- The initial system structure to be used in the second case above.

4.5.2.2 Size of the search space and number of subspaces for the ESP

As in [45], the generation of initial solutions were controlled in a range between 0 and 6. The total number of different solutions to be examined and the number of subspaces are simply given by the following equations:

$$\text{Size of the search space} = \prod_{\gamma=1}^{\Gamma} \prod_{i=1}^{n} Max(J_{\gamma i}) Max(H_{\gamma i}). \tag{4.31}$$

$$\text{Number of subspaces} = \sum_{\gamma=1}^{\Gamma} \sum_{i=1}^{n} Max(J_{\gamma i}) + Max(H_{\gamma i}) + 1. \tag{4.32}$$

Let us consider that different types of elements are allowed to reside in parallel, and assume that $((H_{\gamma i}) = 6\ \forall\gamma,\ 1 \leq \gamma \leq \Gamma$, and $\forall i,\ 1 \leq i \leq n)$. This means simply that a maximum number of 8 elements are allowed for each component. In this case, the search space size is $3, 25.6^{25}.10^{18} \approx 10^{38}$, while the number of subspaces is 296.

4.5.2.3 Parameter settings for the ESP

To avoid parameter overfitting, a procedure similar to that used for the RAP was used again. That is, preliminary numerical tests were used to set the values of the ESP parameters. Different data are randomly generated and used to calibrate the parameters. Once the values of the parameters are set for these preliminary data, they are used for the problem instances to be solved. The parameters' values are: $N_s = 500$, $N_{rep} = 200$, $N_c = 100$, $mnli = 200$, $q = 500$.

4.5.2.4 Comparing the best solutions of SP/TG and existing methods for the ESP

While a large number of methods have been proposed for the RAP, the only existing methods to solve the ESP are a genetic algorithm (GA) in [45] and an ant colony

optimization algorithm in [68]. However, the best-known results are those obtained by the genetic algorithm in [45]. Therefore, the performance of the SP/TG heuristic applied to the ESP is compared with this GA. The system availability of the final solution is rounded to three digits after the decimal point, in order to compare our results with those obtained in [45]. Ten runs of SP/TG algorithm were made using different random seeds for each problem instance. Tables 4.5 and 4.6 show the best expansion plans (among the ten runs) obtained for $Str_0 = Str_0^*$ and $Str_0 = \emptyset$ respectively, with three different desired values of A_0 ($A_0 = 0.950$, $A_0 = 0.970$ and $A_0 = 0.990$). Cost figures are in millions of dollars. The interest rate ρ is chosen to be 0.1. Expansion of each component i at stage γ is presented in the form $X_{i\gamma}(J_{i\gamma})$. Table 4.7 gives a comparison between the best solutions of SP/TG and the best solutions obtained with GA in [45], where the number of runs used is 100 and the best solution over these runs are given. By executing our heuristic over only 10 trials, the obtained results should be *a fortiori* conclusive. The percent that one solution improves upon another is defined in terms of objective function as:

$$\text{MPCI} = 100\% \times \frac{\text{(Minimal GA Cost - Minimal SP/TG Cost)}}{\text{Minimal GA Cost}}.$$

In Table 4.7, the best results are in boldface, indicating that SP/TG out-performed the existing approaches. More specifically, Table 4.7 shows that, in terms of the best objective function (minimum over 10 runs), in 5 of 6 test problems, the solutions found by our algorithm SP/TG are better than those found by GA in [45], while in the remaining instance both algorithms returns the same solution.

A_0	i		$\gamma = 1$	$\gamma = 2$	$\gamma = 3$	$\gamma = 4$	$\gamma = 5$
			$C(1) = 0.645$				$C = 4.127$
		A_γ	0.950	0.951	0.959	0.952	0.960
	1			1(4)			
	2		2(5)		1(5)	1(5)	
0.950	3				1(4)		1(2)
	4		1(7)	1(7)			
	5		1(3)			1(4)	
			$C(1) = 3.323$				$C = 6.484$
		A_γ	0.970	0.971	0.986	0.972	0.970

A_0	i		$\gamma=1$	$\gamma=2$	$\gamma=3$	$\gamma=4$	$\gamma=5$
	1		1(6)	1(6)		1(6)	1(6)
	2		1(3)	1(5)		1(5)	2(5)
0.970	3		1(4)		1(2)		
	4		2(7)				
	5		1(3)				1(4)
		$C(1)=5.503$					$C=7.834$
		A_γ	0.991	0.991	0.990	0.996	0.995
	1		1(6)	1(6)	2(6)		
	2		1(5)	2(5)	1(5)		1(5)
0.990	3		1(2)			1(3)	
	4		1(7)	1(7)	1(7)		
	5		1(3)		1(3)		

TAB. 4.5: The best solution found by SP/TG algorithm among the 10 runs ($Str_0 = Str_0^*$)

A_0	i		$\gamma=1$	$\gamma=2$	$\gamma=3$	$\gamma=4$	$\gamma=5$
		$C(1)=10.160$					$C=14.578$
		A_γ	0.952	0.960	0.951	0.951	0.950
	1		3(4)				2(6)
	2		2(3)	1(5)	1(5)	1(3)	1(5)
0.950	3		1(1)	1(2)			
	4		3(7)	1(9)		1(9)	1(7)
	5		2(3)	1(3)		1(3)	
		$C(1)=12.727$					$C=16.909$
		A_γ	0.970	0.971	0.970	0.996	0.971
	1		3(4)				
	2		3(3)		2(5)		1(5)
0.970	3		2(2)	1(4)		1(2)	
	4		4(7)		1(9)		
	5		3(3)		1(3)		
		$C(1)=17.285$					$C=19.270$
		A_γ	0.994	0.993	0.990	0.996	0.991
	1		3(4)				

A_0	i	$\gamma = 1$	$\gamma = 2$	$\gamma = 3$	$\gamma = 4$	$\gamma = 5$
	2	2(3)	1(4)	3(5)		
0.990	3	3(2)			1(3)	
	4	3(7)	1(9)	1(7)		
	5	3(3)		1(3)		

TAB. 4.6: The best solution found by SP/TG algorithm among the 10 runs ($Str_0 = \emptyset$)

Str_0	A_0	Minimal SP/TG Cost	Minimal GA Cost	$\%MPCI$
	0.950	**4.127**	**4.127**	00.00
$Str_0 = Str_0^*$	0.970	**6.484**	6.519	00.54
	0.990	**7.834**	7.859	00.32
	0.950	**14.578**	14.606	00.19
$Str_0 = \emptyset$	0.970	**16.909**	17.206	01.72
	0.990	**19.270**	19.350	00.41

TAB. 4.7: Total costs for the proposed and existing algorithms for the ESP

4.5.2.5 Robustness of the SP/TG algorithm for the ESP

To measure the robustness of the SP/TG algorithm when applied to the ESP, the standard deviations and the average minimal cost, over ten runs in each instance, are given in Table 4.8. We remark from this table that for each instance, the standard deviation is low. This confirms that the proposed method is robust.

Str_0	A_0	Min cost	Mean cost	Std dev	Average evaluations
	0.950	4.127	4.135	0.008	1,267,647
$Str_0 = Str_0^*$	0.970	6.484	6.532	0.104	1,614,513
	0.990	7.834	7.862	0.030	1,827,918

	0.950	14.578	14.587	0.006	3,140,892
$Str_0 = \emptyset$	0.970	16.909	17.558	0.219	2,484,106
	0.990	19.270	19.474	0.246	2,937,715

TAB. 4.8: Minimum cost, average cost, standard deviation and average evaluation of the SP/TG algorithm

4.5.2.6 Comparing the computational effort of SP/TG and existing methods for the ESP

For each instance, the average numbers of solutions evaluated by SP/TG over ten runs are given in the last column of Table 4.8. The number of solutions evaluated by the proposed SP/TG is $2,2.10^6$ on average (obtained by averaging the values in the last column of Table 4.8). As it is reported in [45] that the number of solutions generated in GA is 2.10^6, the SP/TG algorithm requires only 10 % more computational effort to get the high quality solutions shown above.

4.5.3 Convergence

The convergence curves of SP/TG were drawn for all the test cases, showing similar behaviour in all these cases. Figures 4.6 and 4.7 show for example these curves for the first instance of each problem ($W_0 = 191$ for the RAP, and $A_0 = 0.950$; $Str_0 = Str_0^*$ for the ESP). We remark that even if the solutions of our algorithm are not good during the first iterations of the search process, they become quickly very good.

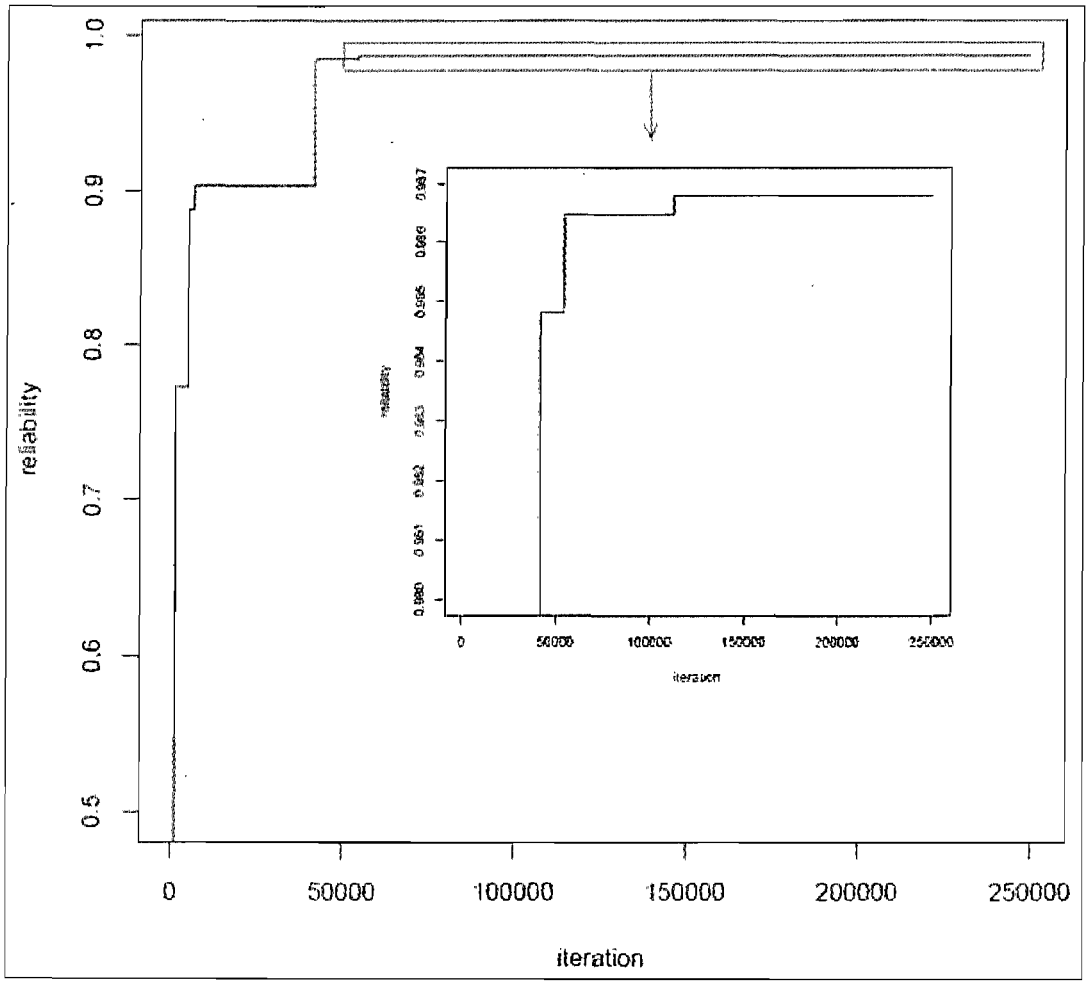


FIG. 4.6 – An example of convergence curve for the RAP

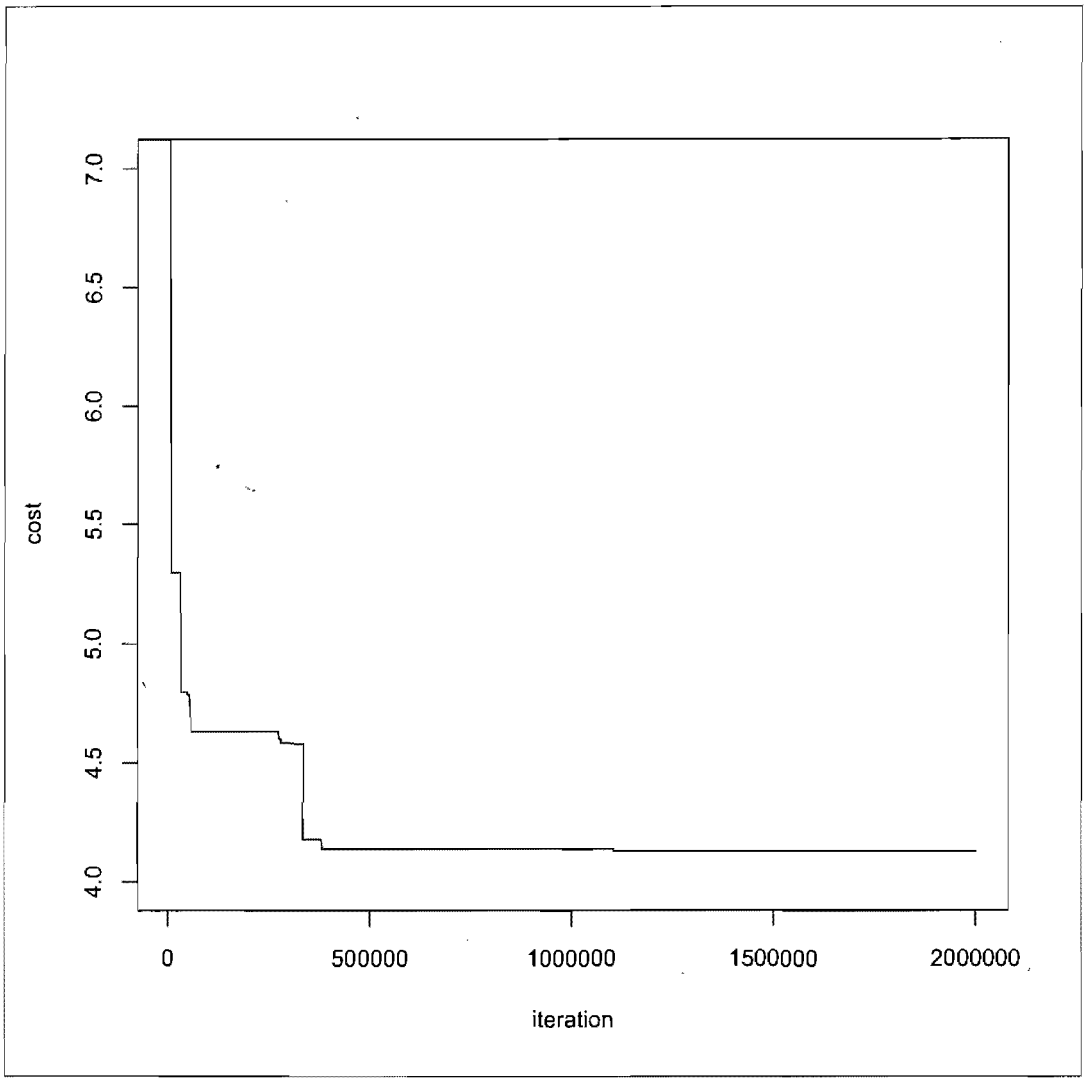


FIG. 4.7 – An example of convergence curve for the ESP

4.6 Conclusion

This paper developed an efficient approach, called SP/TG, to solve two reliability optimization problems for series-parallel systems. The first one is the redundancy allocation problem for series-parallel binary-state systems, and the second one is the expansion-scheduling problem of series-parallel multi-state systems. The newly developed SP/TG approach combines GA, TS and the idea of partitioning the search space. After an appropriate division of the search space into a set of disjoint subspaces, the

role of GA is to select the subspaces, while TS is applied to each selected subspace. By combining two meta-heuristics, the SP/TG provides a balance between diversification and intensification. For both problems, the experimental results showed that the solutions found by the SP/TG approach are better than or are comparable with the best-published results from the literature. As this hybrid approach has been successful for solving two typical design optimization problems from binary-state and multi-state reliability, it may represent a general approach to solve other reliability optimization problems where it is possible to properly partition the search space. Current works concern the application of the SP/TG approach developed in this paper to other reliability optimization problems.

Acknowledgements

The authors would like to thank the editor, and the anonymous reviewers for their comments and helpful questions. They would like to thank also the Natural Sciences and Engineering Research Council of Canada (NSERC) for financial support.

Chapitre 5

Conclusion générale

L'objectif principal de cette thèse a été de développer des algorithmes efficaces pour résoudre des problèmes de conception optimale des systèmes de production, en tenant compte du fait que les composantes sont assujetties à des défaillances aléatoires. Les systèmes étudiés utilisent la redondance comme technique d'amélioration de la performance. La méthodologie poursuivie pour atteindre les objectifs de cette thèse repose sur deux étapes complémentaires :

1. Étape d'évaluation : afin d'estimer la fiabilité d'un système multi-états série-parallèle, on utilise une méthode basée sur la fonction génératrice universelle. Ce choix est motivé par le fait que cette méthode est assez rapide pour être utilisée pour des problèmes d'allocation optimale de la redondance de systèmes multi-états de grandes tailles.
2. Étape d'optimisation : en général, la conception optimale des systèmes de production passe par la formulation de problèmes d'optimisation combinatoire difficiles. Cette thèse a consisté à utiliser des métaheuristiques comme méthodes de résolution pour atteindre le plus rapidement possible une solution, optimale ou quasi-optimale. Les problèmes de conception optimale suivants ont ainsi été considérés :
   (a) Le problème d'allocation de la redondance (PAR) des systèmes séries-parallèles binaires.

(b) Le PAR des systèmes séries-parallèles multi-états homogènes.

(c) Le PAR des systèmes séries-parallèles multi-états hétérogènes.

(d) Le problème de la planification des extensions des systèmes séries-parallèles multi-états.

Ces quatre problèmes sont d'une importance cruciale dans la conception des systèmes de production. Durant ces dernières années, plusieurs métaheuristiques (algorithmes génétiques, recherche avec tabous, algorithmes à colonies de fourmis, ...) ont prouvé leur efficacité pour la résolution de cette famille de problèmes. Dans cette thèse, nous proposons de nouveaux algorithmes en se basant sur une idée originale de partitionnement de l'espace global de recherche en sous-espaces disjoints et sur une approche hybride (combinaison de deux métaheuristiques), afin de tirer profit des avantages des différentes métaheuristiques. Pour chacun des quatre problèmes, nous résolvons les instances existantes dans la littérature, et nous proposons éventuellement des instances de plus grandes tailles. Notre objectif a été bien entendu de prouver la supériorité de nos algorithmes par rapport à ceux existants dans la littérature actuelle.

Tout d'abord, nous avons proposé un algorithme de recherche tabou appliqué à plusieurs sous-espaces de recherche disjoints pour résoudre le problème d'allocation de la redondance, pour un système multi-états série-parallèle homogène. Les résultats numériques obtenus pour trois benchmarks proposés dans la littérature et un quatrième benchmark de taille très grande généré aléatoirement, montrent que l'algorithme proposé présente des avantages, tant au niveau du temps d'exécution qu'au niveau de la qualité des solutions obtenues.

Ensuite, nous avons amélioré le premier algorithme en ajoutant l'algorithme génétique pour sélectionner les sous-espaces de recherche. Il s'agit spécifiquement d'une hybridation de deux métaheuristiques, l'algorithme de recherche avec tabous et l'algorithme génétique. L'algorithme hybride ainsi obtenu combine deux avantages complémentaires : l'algorithme génétique explore des régions plus importantes dans l'espace global de recherche ; le rôle de recherche avec tabous est d'intensifier la recherche de l'optimum global dans les régions qui semblent les plus intéressantes. Cet algorithme est utilisé

pour résoudre le problème d'allocation de la redondance dans un système multi-états série-parallèle, dans le cas non homogène. Différentes expérimentations ont été réalisées sur différents benchmarks. La comparaison entre les résultats obtenus par l'algorithme génétique et les résultats obtenus par l'algorithme hybride ont montré clairement que notre algorithme est plus efficace que les approches existantes pour ce type de problème.

Dans une autre contribution de cette thèse, nous avons appliqué la méthodologie du deuxième article pour résoudre deux autres problèmes difficiles : le problème d'allocation de la redondance des systèmes série-parallèles binaires, et le problème de la planification optimale des extensions pour des systèmes multi-états. Les résultats numériques obtenus pour une quarantaine d'instances, existants dans la littérature, confirment que l'algorithme proposé est très compétitif par rapport aux approches existantes.

Différentes perspectives apparaissent suite à ce travail. Tout d'abord, la stratégie de partitionnement de l'espace global de recherche pourrait sans doute être rendue plus efficace si on ajoutait des phases de diversification de la recherche dans chaque sous-espace de recherche. Cela permettrait d'améliorer la qualité de la solution. Le prix à payer pour cette qualité supérieure est l'augmentation du temps de résolution. De plus, il serait intéressant de paralléliser nos approches puisque les sous-espaces sont disjoints. De même, les concepts proposés dans cette thèse sont applicables à d'autres types de métaheuristiques basées sur une recherche locale. Ceci pourrait, par exemple, donner lieu à une hybridation entre plusieurs méthodes de résolution, et pourrait sans doute améliorer les résultats. Une amélioration des techniques utilisées pour le calibrage des paramètres des algorithmes serait aussi utile.

Au niveau du calcul de performance des systèmes multi-états, il serait intéressant de développer une nouvelle méthode d'évaluation de disponibilité/fiabilité et étendre la méthodologie proposée pour d'autres types de systèmes qui ne présentent pas une structure série-parallèle ou parallèle-série, ainsi que pour ceux dont les composantes ne sont pas indépendantes.

Une suite logique à nos travaux serait d'intégrer des aspects de la maintenance corrective et de la maintenance préventive dans la conception optimale des systèmes

multi-états séries-parallèles. Comme autres pistes de recherche, il serait intéressant de résoudre le problème de la planification optimale des extensions pour des systèmes multi-états, dans lequel les composantes ajoutées à chaque période ne sont pas forcément de même version (comme c'est le cas dans le chapitre 4). Finalement, il serait également intéressant d'étendre la méthodologie proposée à d'autres problèmes d'optimisation combinatoire pour lesquels il est possible de diviser l'espace global de recherche.

Bibliographie

[1] Agarwal, M. et R. Gupta. 2007. Homogeneous redundancy optimization in multi-state series-parallel systems : A heuristic approach. *IIE Transactions* **39(3)** 277–289.

[2] Ait-Kadi, D. et M. Nourelfath. 2001. Availability optimization of fault-tolerant systems. *Int Conf on Ind Engng and Prod Manage (IEPM'2001)*. Québec, Canada.

[3] Anderson, T. et P.A. Lee. 1981. *Fault Tolerance : Principles and Practice*. Prentice Hall International, Englewood Cliffs N.J.

[4] Arlat, J., D. Essame et D. Powell. 2000. Tolérance aux fautes dans les systèmes critiques. Tech. Rep. 151, Rapport LAAS-CNRS.

[5] Aven, T. 1993. On performance measures for multistate monotone systems. *Reliability Engineering and System Safety* **41** 259–266.

[6] Barlow, R.E. et F. Proschan. 1981. *Statistical theory of reliability and life testing : probability models*. Silver Spring, MD.

[7] Barlow, R.E. et A.S. Wu. 1978. Coherent systems with multi-state components. *Mathematics of Operations Research* **3** 275–281.

[8] Bellman, R. et S.E. Dreyfus. 1958. Dynamic programming and reliability of multicomponent devices. *Operation Research* **6** 200–206.

[9] Billinton, R. et R. Allan. 1984. *Reliability of power systems*. Pitman.

[10] Billinton, R. et R. Allan. 1990. *Reliability evaluation of power systems*. Pitman.

[11] Bris, R., E. Châtelet et F. Yalaoui. 2003. New method to minimize the preventive maintenance cost of series-parallel systems. *Reliability Engineering and System Safety* **82(3)** 247–255.

[12] Bulfin, R.L. et C.Y. Liu. 1985. Optimal allocation of redundant components for large systems. *IEEE Tran. Reliab.* **R-34** 241–247.

[13] Cafaro, G., F. Corsi et F. Vacca. 1986. Multistate markov models and structural properties of the transition rate matrix. *IEEE Transactions on Reliability* **R-35** 192–200.

[14] Chern, M.S. 1992. On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letters* **11** 309–315.

[15] Coit, D.W. et A.E. Smith. 1996. Penalty guided genetic search for reliability design optimization. *Computers & Industrial Engineering* **30(4)** 895–904.

[16] Coit, D.W. et A.E. Smith. 1996. Reliability optimization of series-parallel systems using a genetic algorithm. *IEEE Transactions on Reliability* **45(2)** 254–260.

[17] Dhillon, B. 1986. Reliability apportionment/allocation : a survey. *Microelectronics and Reliability* **26(6)** 1121–1129.

[18] Dimitrov, B., V. Rykov et P. Stanchev. 2002. On multi-state reliability systems. 3^{rd} *International Conference on mathematical Methods in Reliability (MMR 2002).* 201–204.

[19] El-Neweihi, E., F. Proschan et J. Sethuraman. 1978. Multistate coherent systems. *Journal of Applied Probability* **15(12)** 675–688.

[20] Fyffe, D.E., W.W. Hines et N.K. Lee. 1968. System reliability allocation and a computational algorithm. *IEEE Transactions on Reliability* **R-17** 64–69.

[21] Gen, M., K. Ida, Y. Tsujimura et C. E. Kim. 1993. Large scale 0-1 fuzzy goal programming and its application to reliability optimization problem. *Computers & Industrial engineering* **24** 539–549.

[22] Gendreau, M. 2002. *Recent Advances in Tabu Search, in Essays and Surveys in Metaheuristics, C.C. Ribeiro and P.Hansen(eds.).* Kluwer Academic Publishers.

[23] Gendreau, M., F. Guertin et J.Y. Potvin, E. Taillard. 1999. parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science* **33** 381–390.

[24] Gendreau, M., A. Hertz et G. A. Laporte. 1994. tabu search heuristic for the vehicle routing problems. *Management Science* **40** 1276–1290.

[25] Gendreau, M. et J.Y. Potvin. 2003. A guide to tabu search. *Technical Report, CRT-2003-23,* Centre de Recherche sur les Transports, Université de Montréal.

[26] Gendreau, M., P. Soriano et L. Salvail. 1993. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research* **41** 385–403.

[27] Ghare, M. et R.E. Taylor. 1969. Optimal redundancy for reliability in series system. *Oper Res* **17** 838–847.

[28] Glover, F. 1977. Heuristics for integer programming using surrogate constraints. *Decision Sciences* **8** 156–166.

[29] Glover, F. 1986. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research* **13** 533–549.

[30] Glover, F. 1989. Tabu Search - part I. *ORSA Journal on Computing* **1** 190–206.

[31] Glover, F. 1990. Tabu Search - part II. *ORSA Journal on Computing* **2** 4–32.

[32] Glover, F. et M. Laguna. 1993. *Tabu Search : A Chapter in Modern Heuristic Techniques for Combinatorial Problems, Reeves c., Blackwell Scientific.* Blackwell, Oxford.

[33] Glover, F. et M. Laguna. 1997. *Tabu Search.* Kluwer Academic Publishers.

[34] Glover, F., E. Taillard et D. de Werra. 1993. A user's guide to tabu search. *Annals of Operations Research* **41** 13–28.

[35] Hsieh, Y.C. 2002. A linear approximation for redundant reliability problems with multiple component choices. *Computers & Industrial engineering* **44** 91–103.

[36] Goldberg, D. 1989. *Genetic Algorithm In Search, Optimization And Machine Learning*. Addison-Wesley.

[37] Holland, J. H. 1975. *Adaptation In Natural And Artificial Systems*. University of Michigan Press.

[38] Huang, J. et M.J. Zuo. 2000. Generalized multi-state k-out-of n : G systems. *IEEE Transactions on Reliability* **49(1)** 105–111.

[39] Hwang, C.L., F.A. Tillman et W. Kuo. 1979. Reliability optimization by generalized lagrangian-function and reduced-gradient methods. *IEEE Tran. Reliab.* **R28(4)** 316–319.

[40] Kelly, P.J., M. Laguna et F. Glover. 1994. A study of diversification strategies for the quadratic assignment problem. *Computers and Operations Research* **21(8)** 885–893.

[41] Kulturel-Konak, S., A.E. Smith et D.W. Coit. 2003. Efficiently solving the redundancy allocation problem using tabu search. *IIE transactions* **35** 515–526.

[42] Kuo, W. et VR. Prasad. 2000. An annotated overview of system-reliability optimization. *IEEE Transactions on Reliability* **49(2)** 176–187.

[43] Kuo, W., V.R. Prasad, F.A. Tillman et C.L. Hwang. 2001. *Optimal Reliability Design : Fundamentals and applications*. Cambridge University Press.

[44] Lam, C.T. et R.H. Yeh. 1994. Optimal maintenance policies for deteriorating systems undervarious maintenance strategies. *IEEE Transactions on Reliability* **43(3)** 423–430.

[45] Levitin, G. 2000. Multistate series-parallel system expansion-scheduling subject to availability constraints. *IEEE Transactions on reliability* **49(1)** 71–79.

[46] Levitin, G. 2005. *Universal generating function in reliability analysis and optimization*. Springer-Verlag.

[47] Levitin, G. 2006. Private communication : Email from dr. g. levitin. November 26th, 2006.

[48] Levitin, G. et A. Lisnianski. 2000. Optimal replacement scheduling in multi-state series-parallel systems, vol 16, 2000. *Quality and Reliability Engineering International* **16** 175–162.

[49] Levitin, G. et A. Lisnianski. 2001. A new approach to solving problems of multi-state system reliability optimization. *Quality and Reliability Engineering International* **47(2)** 93–104.

[50] Levitin, G., A. Lisnianski, H. Ben-Haim et D. Elmakis. 1997. Structure optimization of power system with different redundant elements. *Electric Power Systems Research* **43(1)** 19–27.

[51] Levitin, G., A. Lisnianski, H. Ben-Haim et D. Elmakis. 1998. Redundancy optimization for series-parallel multi-state systems. *IEEE Transactions on reliability* **47(2)** 165–172.

[52] Levitin, G., A. Lisnianski, H. Ben-Haim et D. Elmakis. 2000. Genetic algorithm and universal generating function technique for solving problems of power system reliability optimization. *International Conference on Proceedings DRPT*. 582–586.

[53] Liang, Y.C. et Y.C. Chen. 2007. Redundancy allocation of series-parallel systems using a variable neighborhood search algorithm. *IEEE Transactions on reliability* **92** 323–331.

[54] Liang, Y.C. et A.E. Smith. 2004. An ant colony optimization algorithm for the redundancy allocation problem. *IEEE Transactions on reliability* **53(3)** 417–423.

[55] Lisnianski, A. et G. Levitin. 2003. *Multi-state system reliability. Assesment, optimization and applications*. World scientific.

[56] Lisnianski, A., G. Levitin, H. Ben-Haim et D. Elmakis. 1996. Power system structure optimization subject to reliability constraints. *Electric Power Systems Research* **39(2)** 145–152.

[57] Liu, Y.W. et K.C. Kapur. 2006. Reliability measures for dynamic multistate non-repairable systems and their applications to system performance evaluation. *IIE Transactions* **38(6)** 511–520.

[58] Misra, K. 1986. On optimal reliability design : a review. *System Science* **12** 5–30.

[59] Misra, K.B. et U. Sharma. 1991. An efficient algorithm to solve integer-programming problems arising in system-reliability design. *IEEE Trans Reliab* **40(1)** 81–91.

[60] Murchland, J. 1975. *Fundamental concepts and relations for reliability analysis of multi-state systems, Reliability and Fault Tree Analysis*. Philadelphia: ed. R. Barlow, J. Fussell.

[61] Nahas, N. et M. Nourelfath. 2005. Ant system for reliability optimization of a series system with multiple-choice and budget constraints. *Reliability Engineering and System Safety* **87(1)** 1–12.

[62] Nahas, N., M. Nourelfath et Ait-Kadi D. 2007. Coupling ant colony and the degraded ceiling algorithm for the redundancy allocation problem of series-parallel systems. *Reliability Engineering and System Safety* **92(2)** 211–222.

[63] Nakagawa, Y. et S. Miyazaki. 1981. Surrogate constraints algorithm for reliability optimization problems with two constraints. *IEEE Transactions on Reliability* **R-30** 175–180.

[64] Niel, É. 1994. De la sécurité opérationnelle des systèmes de production. Tech. Rep. Déc., Rapport technique, INP Grenoble, France.

[65] Nourelfath, M. 1997. Extension de la théorie de la supervision à la surveillance, à la commande des systèmes à événements discrets. Ph.D. thesis, Thèse de doctorat, INSA Lyon, France.

[66] Nourelfath, M., D. Ait-Kadi et Soro I.W. 2003. Availability modeling and optimization of reconfigurable manufacturing systems. *Journal of Quality in Maintenance Engineering.* **9(3)** 284–302.

[67] Nourelfath, M. et A. Khatab. 2004. Ant colony optimization for multi-state series-parallel system expansion-scheduling. *7th International Workshop on discrete Event Systems (WODES'04)*. Reims 22-24 September, 259–264.

[68] Nourelfath, M., N. Nahas et A. Zeblah. 2003. An ant colony approach to redundancy optimization for multi-state system. *Int Conf on Ind Engng and Prod Manage (IEPM)*. Porto-Portugal.

[69] Osman, I.H. et G. Laporte. 1996. Metaheuristics : a bibliography. *Annals of Operations Research* **63** 513–623.

[70] Ouzineb, M., M. Nourelfath et M. Gendreau. May 2009. An efficient heuristic for reliability design optimization problems. *Computers & Operations Research*, à paraître 2009.

[71] Ouzineb, M., M. Nourelfath et Gendreau M. 2008. Tabu search for the redundancy allocation problem of homogenous series parallel multi-state systems. *Reliability Engineering and System Safety* **93(8)** 1257–1272.

[72] Pham, H., A. Suprasad et R.B. Misra. 1997. Availability and mean life time prediction of multistage degraded system with partial repairs. *Reliability Engineering and System Safety* **56(2)** 169–173.

[73] Ramirez, M. et D.W. Coit. 2004. A heuristic for solving the redundancy allocation problem for multi-state series-parallel systems. *Reliability Engineering and System Safety* **83** 341–349.

[74] Ross, S. 1979. Multivalued state component systems. *Annals of Probability* **7** 379–383.

[75] Ross, S. 1993. *Introduction to probability models*. Academic press.

[76] Samrout, M., F. Yalaoui, E. Châtelet et N. Chebbo. 2005. New methods to minimize the preventive maintenance cost of series-parallel systems using ant colony optimization. *Reliability Engineering and System Safety* **89(3)** 346–354.

[77] Tillman, F.A., C.L. Hwang et W. Kuo. 1977. Optimization of systems reliability with redundancy a review. *IEEE Transactions on Reliability* **26(3)** 148–155.

[78] Tillman, F.A., C.L. Hwang et Kuo W. 1980. *Optimization of systems reliability*. Marcel Dekker, NY.

[79] Tzafestas, S.G. 1980. Optimization of system reliability : A survey of problems and techniques. *International Journal of System Science* **11(4)** 455–486.

[80] Ushakov, I. 1986. Universal generating function. *Sov. J. Computing System Science* **24(5)** 118–129.

[81] Ushakov, I. 1987. Optimal standby problems and a universal generating function. *Sov J Computing System Science* **25(4)** 79–82.

[82] Ushakov, I., G. Levitin et A. Lisnianski. 2002. Multi-state system reliability: from theory to practice. *Proc. of 3 Int Conf on mathematical methods in reliability (MMR)*. Trondheim, Norway, 635–638.

[83] Villemeur, A. 1988. *Sûreté de fonctionnement des systèmes industriels*. Édition Eyrolles, Paris.

[84] Whitley, D. et J. Kauth. 1988. Genitor : A different genetic algorithm. Tech. Rep. CS-88-101, Colorado State University.

[85] Wu, S. et L.Y. Chan. 2003. Performance utility-analysis of multi-state systems. *IEEE Transactions on Reliability* **52(1)** 14–19.

[86] Xue, J. et K. Yang. 1995. Dynamic reliability analysis of coherent multi-state systems. *IEEE Transactions on Reliability* **44** 683–688.

[87] Yalaoui, A. 2004. Allocation de fiabilité et de redondance dans les systèmes parallèle-série et série-parallèle. Ph.D. thesis, Thèse de doctorat, UTT Troyes, France.

[88] Yalaoui, A., E. Châtelet et C. Chu. 2005. A new dynamic programming method for reliability and redundancy allocation in a parallel-series system. *IEEE Tran. Reliab.* **54** 254–261.

[89] Yu, H., F. Yalaoui, E. Châtelet et C. Chu. 2007. Optimally desiging a cold-standby system with certain maintenance policy. *Reliability Engineering and System Safety* **92(1)** 85–91.

[90] Zio, E. et L. Podofillini. 2004. A Monte Carlo approach to the estimation of importance measures of multi-state components. *Reliability and Maintainability Annual Symposium* 129–134.